tanulmányok

137/1982

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

Д О К Л А Д Ы С И М П О З И У М О В

проводимых 23-28 апреля 1979 года в Вайсиге и
5-10 мая 1980 года в Вишеграде
по время совещаний НКС СЭВ по теме 1-15.1
"ТЕОРИЯ АВТОМАТОВ И ЕЕ ПРИЛОЖЕНИЯ К ПРОЕКТИРОВАНИЮ
ДИСКРЕТНЫХ УСТРОЙСТВ И СИСТЕМ"

Tanulmányok 137/1982

A kiadásért felelős:

VÁMOS TIBOR

Szerkesztette:

IVICS JOZSEF

ISBN 963 311 145 5
ISSN 0324 2951

ELŐSZÓ

A *KGST* keretében működő "Automataelmélet kidolgozása és alkalmazása a diszkrét rendszerek tervezésében". *1–15.1 szakbizottság* 1979. április 23–28. között Weissigben az NDK Tudományos Akadémia Központi Kibernetikai Intézete, 1980. május 5–10. között Visegrádon az MTA Számítástechnikai és Automatizálási Kutató Intézete szervezésében tartotta ülését. A szakbizottsági ülések a résztvevő országok munkabeszámolóinak és a következő időszakra vonatkozó munkatervének a megvitatása után *szimpóziumként* folytatódik, ahol a témához kapcsolódó, az egyes országokban elért legújabb eredményekről és a jövőbeni elképzelésekről előadások hangzottak el. Az eddigi gyakorlat szerint az *elhangzott előadásokról* kiadvány készült az MTA SZTAKI gondozásában.
Jelen kiadvány a negyedik és egyben az utolsó.

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ

Глава 1.: ДОКЛАДЫ СИМПОЗИУМА НКС 1979 ГОДА

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

АНДРЕЙ КРАСНЕВСКИ

ПОВЕДЕНИЕ АВТОМАТОВ С ПЕРЕМЕННОЙ СТРУКТУРОЙ В УСЛОВНОЙ СЛУЧАЙНОЙ СРЕДЕ

Политехнический Институт, Варшава

ВВЕДЕНИЕ

Для многих систем динамического управления, в которых множество значений управляющих переменных конечное и информация о поведении процесса передаваемая устройству управления дискретная, задача синтеза оптимального алгоритма управления эквивалентна исследованной в кибернетике задаче оптимального функционирования автомата с переменной структурой в случайной среде.

В работах [1], [2], [5] предложен, например, метод управления, в котором коммутируемая сеть связи с изменяющимися во времени параметрами /потоки передаваемой информации, тяготения, ёмкости ветвей, возможность выхода из строя ветвей и узлов коммутации/ рассматривается в качестве случайной среды, а управляющая система - в качестве коллектива автоматов с переменной структурой взаимодействующих с этой средой /рис. 1/.
Задачу оптимального управления распределением информации в сети связи можно тогда решать методами теории оптимального поведения автомата в случайной среде.

В литературе особенно много результатов в области поведения автоматов в стационарной случайной среде, но стационарная среда как модель сети связи обладает многими недостатками /нет, например, возможности моделирования изменяющихся по времени параметров сети/ так, что возникает необходимость построения более точных моделей. Многими преимуществами обладает модель в виде нестационарной случайной среды с марковской цепью для описания вероятностей штрафа [3].

В случае сети связи, в которой распределением потоков информации управляет децентрализованная система с альтернативным выбором транзитного узла для исследования эффективности алгоритмов управления надо построить модель в виде нестационарной условной случайной среды.

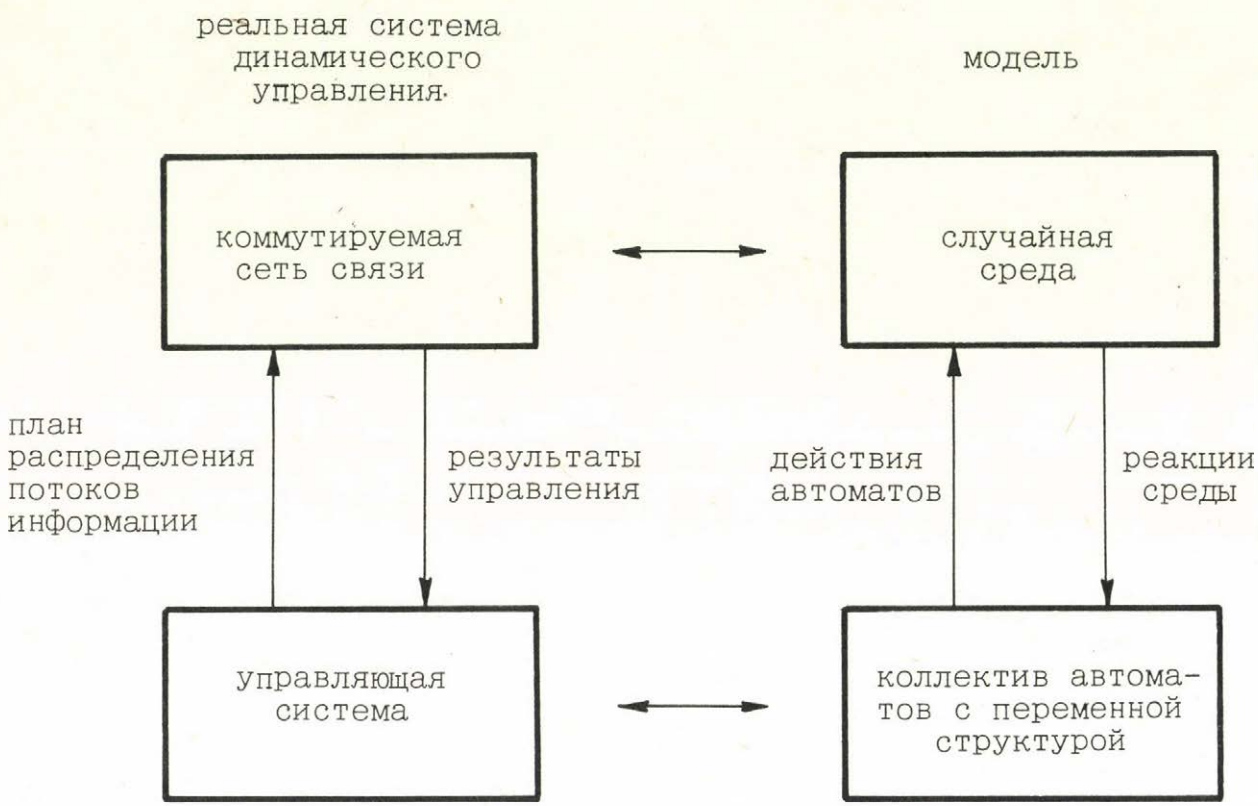


Рис. 1. Модель системы управления сетью связи

1. НЕСТАЦИОНАРНАЯ УСЛОВНАЯ СЛУЧАЙНАЯ СРЕДА

Определим сперва формально понятие условной случайной среды. Условная случайная среда /УСС/ задается как совокупность

$$E = \langle Y, y_o, X, p, c \rangle$$

где: $Y = \{y_1, y_2, \ldots, y_n\}$ – множество допустимых входных действий среды

y_o – праздное действие

$X = \{0,1\}$ – множество реакции среды

$p: Y \to \{0,1\}$ - вектор вероятностей приема входных действий

$c: Y \to \{0,1\}$ - вектор вероятностей штрафа для входных действий

Сформируем дополнительно множество эффективных действий УСС

$$V = Y \cup \{y_o\}$$

В каждый момент времени τ эффективное действие среды v^τ определяется в зависимости от входного действия y^τ и предшествующего эффективного действия $v^{\tau-1}$ случайным образом по следующей формуле:

$v^o = y_o$

если $v^{\tau-1} \neq y_o$, то $v^\tau = v^{\tau-1}$

если $v^{\tau-1} \neq y_o$, то $P\{v^\tau = y^\tau\} = p^\tau$

$$P\{v^\tau = y_o\} = 1 - p^\tau, \text{ где } p^\tau = p_i \Leftrightarrow y^\tau = y_i$$

Реакция УСС на эффективное действие в виде штрафа ($x^\tau = 1$) или поощрения ($x^\tau = 0$) производится случайно по вектору вероятностей штрафа

$$P\{x^\tau = 1\} = \begin{cases} c^\tau, & \text{если } v^\tau \neq y_o \\ 1, & \text{если } v^\tau = y_o \end{cases}$$

$$P\{x^\tau = 0\} = \begin{cases} 1 - c^\tau, & v^\tau \neq y_o \\ 0, & v^\tau = y_o \end{cases}$$

Рассмотрим поведение УСС под влиянием ряда входных действий /входного слова/ $y^* \in Y^*$, где $Y^* = \{y^1 y^2 \ldots y^r \mid y^i \in Y, y^i \neq y^j\}$ - множество всех входных слов состоящих из различных входных действий.

Для входного слова $y^* = y^1 y^2 \ldots y^r \in Y^*$ эффективное действие среды v определенное вектором p.

$$P\{v = y^i\} = p^i \prod_{j=1}^{i-1} (1 - p^j), \quad i = 1,2,\ldots,r$$

$$P\{v = y_o\} = \prod_{j=1}^{r} (1 - p^j)$$

а реакция среды **x** - вектором **c**

$$P\{x = 1 \mid v = y_i\} = c_i$$

$$P\{x = 0 \mid v = y_i\} = 1 - c_i$$

/для праздного действия вероятность штрафа c_o равна 1/.

Очередным словам поступающим на вход УСС поставим в соответствие очередные натуральные числа $k = 1,2,...$ Предположим, что для очередных слов $y^*(k)$, $k = 1,2,...$, на входе среды векторы **p** и **c** отличаются по значению в этом случае говорим, что среда - нестационарная.

Нестационарная случайная среда с дискретным временем /НУСС/ определяется формально как

$$E_k = \langle Y, y_o, X, p, c \rangle$$

где: Y, y_o, X - как для УСС

$p: \mathcal{N} \to P_y = \{f \mid f: Y \to [0,1]\}$, $c: \mathcal{N} \to P_y$

$p(k): Y \to \{0,1\}$ - вектор вероятностей приема входных действий в момент k.

$c(k): Y \to \{0,1\}$ - вектор вероятностей штрафа для входных действий в момент k.

Схема функционирования НУСС представлена на рис. 2.

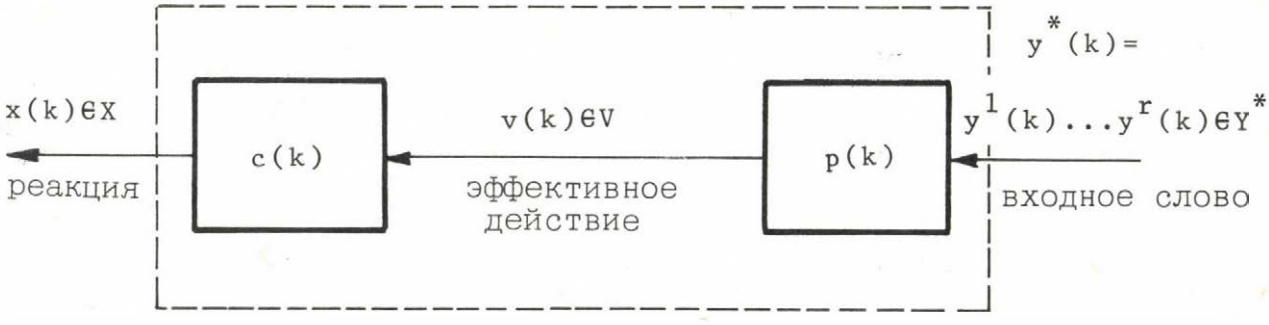


Рис. 2 . Нестационарная условная случайная среда

НУСС можно использовать как модель многих реальных объектов, особенно сетей связи.

2. НУСС КАК МОДЕЛЬ СЕТИ СВЯЗИ

Рассмотрим сеть связи состоящая из N узлов $y_1, y_2, \ldots, y_N$ и поток вызовов на составление соединения от узда y_i к узлу y_j. Соединения от y_i к y_j могут устанавливаться по направлениям /транзитным узлам/ $y_1^{ij}, y_2^{ij}, \ldots, y_n^{ij}$ /рис. 3/.

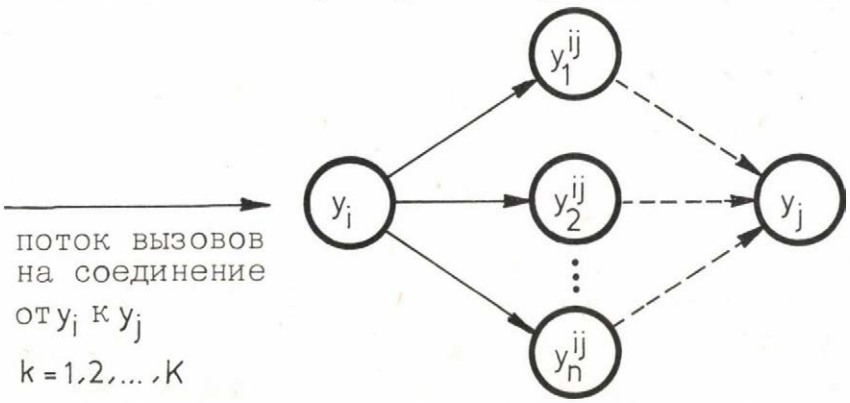


Рис. 3. Процесс составления соединения в сети связи

Процессом составления соединения управляет децентрализованный алгоритм альтернативного выбора транзитного узла.

Пусть вызовы поступают в дискретные моменты времени $k = 1,2,\ldots, K$. Для каждого вызова очередно проверяется возможность его подключения к транзитным узлам пока не найдется направления со свободным каналом. Если транзитный узел является узлом назначения y_j, устанавливается соединение с требуемым абонентом узла y_j, в противном случае весь процесс выбора следующего узла повторяется. Если в узле y_i не нашлось ни одного направления со свободным каналом или число транзитов обслуживаемого вызова превысило допустимое, вызов получает отказ /соединение не будет установлено/.

Порядок проверки транзитных направлений определен алгоритмом управления может быть модифицирован в зависимости от изменяющейся ситуации в сети связи.

Представленному процессу составления соединения от y_i к y_j поставим в соответствие формальную модель в виде НУСС.

$$E_k^{ij} = \langle Y^{ij}, y_o, X, p^{ij}, c^{ij} \rangle$$

где: $Y^{ij} = \{y_1^{ij}, y_2^{ij}, \ldots, y_n^{ij}\}$ – множество допустимых транзитных узлов

y_o – дополнительный, фиктивный узел, к которому подключаются вызовы, которые получили отказ /в узле y_i не нашлось свободного канала к какому-нибудь транзитному узлу

$x^{ij}(k) = \begin{cases} 0, & \text{если установилось соединение от } y_i \text{ к } y_j \\ 1, & \text{если вызов получил отказ} \end{cases}$

$p_\ell^{ij}(k)$ – вероятность составления соединения от y_i к транзитному узлу y_ℓ^{ij}

$c_\ell^{ij}(k)$ – вероятность отказа для вызова от транзитного узла y_ℓ^{ij} к узлу назначения y_j

$y^*(k) = y^1(k) y^2(k) \ldots y^n(k)$ – определяет порядок проверки транзитных узлов

$v(k)$ – узел, к которому нашелся свободный канал и производится транзитное подключение вызова.

Задача оптимального управления распределением информации в сети связи /в смысле минимизации числа отказов для рассматриваемого потока вызовов от y_i к y_j / соответствует задаче минимизации величины математического ожидания средней реакции НУСС за данный период времени

$$Q_k^{ij} = E\left\{\frac{1}{K} \sum_{k=1}^{K} x^{ij}(k)\right\}$$

3. УСЛОВИЯ ОПТИМАЛЬНОГО ФУНКЦИОНИРОВАНИЯ НУСС

Рассмотрим функционирование НУСС

$$E_k = \langle Y, y_o, X, p, c \rangle$$

под влиянием последовательности входных слов $y_k^* = \{y^*(k)\}_{k=1,2,\ldots,K}$

Для данного момента времени величина математического ожидания реакции среды зависит от входного слова

$$E\{x(k)\} = E\{x(k) \mid y^*(k)\}, \quad y^*(k) \in Y^*$$

и следовательно величина математического ожидания средней реакции НУСС зависит от ряда входных слов среды.

$$Q_k = E\left\{\frac{1}{K}\sum_{k=1}^{K} x(k)\right\} = \frac{1}{K}\sum_{k=1}^{K} E\{x(k)\} =$$

$$= \frac{1}{K}\sum_{k=1}^{K} E\{x(k) \mid y^*(k)\} = E\left\{\frac{1}{K}\sum_{k=1}^{K} x(k) \mid y_k^*\right\}$$

Можно доказать следующую теорему, которая определяет необходимые и достаточные улсовия для того, чтобы величина достигала минимума.

Теорема 1.

$\hat{y}_k^* = \{\hat{y}^*(k)\}_{k=1,2,\ldots,K}$ является оптимальной последовательностью входных слов для НУСС

$$\forall\, y_k^* \quad Q_k(y_k^*) \geq Q_k(\hat{y}_k^*)$$

тогда и только тогда, если для каждого момента времени $k = 1,2,\ldots,K$ входное слово $y^*(k) = y^1(k)y^2(k)\ldots y^r(k)$ выполняет следующие условия:

а/ $r = n = |y|$

б/ $(\hat{y}^i(k) = y_m) \wedge (\hat{y}^j(k) = y_\ell) \wedge (c_m(k) < c_\ell(k)) \Rightarrow (i < j)$ $\quad i,j=1,2,\ldots,r$

Условие а/ говорит, что оптимальное входное слово должно состоять из всех входных действий среды. Условие б/ определяет порядок действий в оптимальном входном слове /действия, которым соответствует меньшее значение вероятности штрафа должны предшествовать действиям, для которых значение вероятности штрафа большое/.

4. ПОВЕДЕНИЕ АВТОМАТОВ С ПЕРЕМЕННОЙ СТРУКТУРОЙ В НУСС

Принцип взаимодействия НУСС и автомата с переменной структурой /АПС/ представлен на рис. 4.

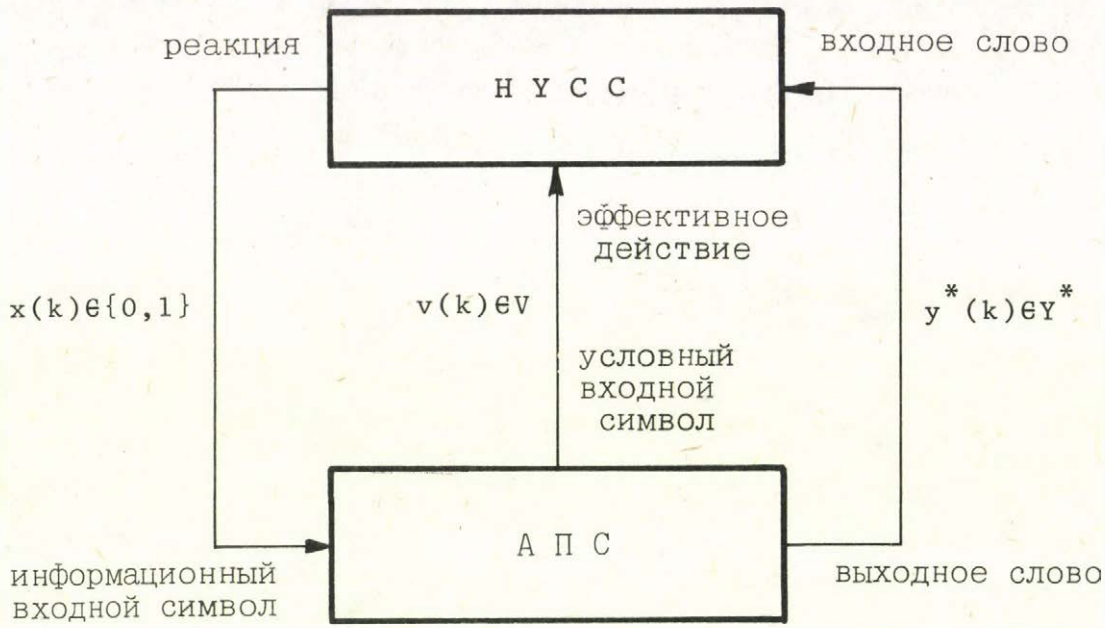


Рис. 4. Взаимодействие АПС с НУСС

Для каждого k (k = 1,2,...,K) АПС взаимодействующий с НУСС формирует свое выходное слово $y^*(k)$ в зависимости от эффективных действий и реакции среды на предшествующие слова.

В качестве АПС функционирующего в случайной среде рассматривается вероятностный или размытый автомат с переменной структурой [1], [4]. Автомат взаимодействующий с НУСС отличается дополнительным входом, на который поступают эффективные действия среды. Такой автомат будем звать условным автоматом с переменной структурой /УАПС/. Определим два типа УАПС - вероятностный УАПС и размытый УАПС.

Вообще условный автомат с переменной структурой задается как совокупность

$$M = \langle A, Y, X, V, \pi^0, \phi \rangle$$

где: $A = \{a_1, a_2, \ldots, a_n\}$ - множество внутренних состояний,

$Y = \{y_1, y_2, \ldots, y_n\}$ - множество выходных символов,

$X = \{0,1\}$- множество информационных входных символов,

$V = Y \cup \{y_o\}$- множество условных входных символов,

π^0 - функция, которая определяет начальное состояние автомата,

ϕ - алгоритм модификации автомата.

Для вероятностного условного автомата с переменной структурой /ВУАПС/

$$\pi^0 \in \mathcal{P}(A) = \{\pi \mid \pi : A \to [0,1],\ \sum_{a_i \in A} \pi_i = 1, \pi > 0\}$$

π^0 - функция вероятности выбора состояний в начальный момент времени.

Алгоритм модификации

$$\phi : \mathcal{P}(A) \times V \times X \to \mathcal{P}(A)$$

определяет метод модификации функции вероятности выбора состояния в зависимости от условного и информационного входа /эффективного действия и реакции НУСС на последнее входное слово автомата/

$$\pi(k+1) = \phi(\pi(k), v(k), x(k)), \quad \pi(1) = \pi^0$$

Функция вероятности выбора состояния $\pi(k)$ определяет последовательность внутренних состояний и следовательно выходное слово ВУАПС по следующей формуле:

$$P\{y^*(k) = y^1(k)\ y^2(k) \ldots y^n(k)\} = \prod_{i=1}^{n} \frac{\pi^i(k)}{1 - \sum_{j=1}^{i-1} \pi^j(k)}$$

где:

$$(y^i(k) = y_\ell) \Rightarrow (\pi^i(k) = \pi_\ell(k))$$

Для размытого условного автомата с переменной структурой /РУАПС/

$$\pi^0 \in \underset{\sim}{\mathcal{F}}(A) = \{\pi \mid \pi : A \to [0,1]\}$$

π^0 - функция принадлежности размытого состояния [6] РУАПС в начальный момент времени.

Алгоритм модификации

$$\phi : \underset{\sim}{\mathcal{F}}(A) \times V \times X \to \underset{\sim}{\mathcal{F}}(A)$$

определяет метод модификации функции принадлежности размытого состояния в зависимости от условного и информационного входа

$$\pi(k+1) = \phi(\pi(k), \quad v(k), x(k)), \qquad \pi(1) = \pi^0$$

Размытое состояние определяет выходное слово ВУАПС по следующей формуле:

$$(\forall\, y_i, y_j \in Y) \quad (y^{\ell}(k) = y_i) \wedge (y^{m}(k) = y_j) \wedge (\pi_i(k) > \pi_j(k)) \Rightarrow (\ell < m)$$

Мерой целесообразности поведения УАПС в НУСС является величина математического ожидания средней реакции среды за данный период времени

$$Q_k = E\left\{\frac{1}{K} \sum_{k=1}^{K} x(k)\right\}$$

Теорема 1 определяет необходимые и достаточные условия для того, чтобы значения Q_k достигло минимума. Таким образом оптимальная последовательность выходных слов УАПС должна выполнять условия представленные теоремой 1.

Можно легко доказать теорему формируемая необходимые и достаточные условия, которым должна отвечать функция принадлежности размытого состояния РУАПС, для того, чтобы поведение РУАПС в НУСС было оптимальным.

Теорема 2

РУАПС взаимодействующий с НУСС формирует оптимальную последовательность входных слов среды $\hat{y}_k^*$ тогда и только тогда, если для каждого момента времени $k = 1,2,\ldots,K$ функция принадлежности размытого состояния РУАПС выполняет условие

$$(c_i(k) < c_j(k)) \Rightarrow (\pi_i(k) > \pi_j(k))$$

Условие определенное теоремой 2 не имеет большого практического значения, потому что рассматривая проблему поведения УАПС в НУСС предполагаем, что вектор вероятностей штрафа НУСС неизвестный автоматом. Таким образом, функция принадлежности раз-

мытого состояния РУАПС и функция вероятности выбора состояния ВУАПС формируется алгоритмом модификации УАПС в зависимости от реакции среды за эффективные действия.

Для ВУАПС взаимодействующего с НУСС невозможно сформировать хотя бы теоретические условия оптимальности /для РУАПС они определены теоремой 2/, более того можно доказать следующую теорему:

Теорема 3

Для каждой последовательности $\pi_k = \{\pi(k)\}_{k=1,2,\ldots,K}$ вероятностей выбора состояния ВУАПС взаимодействующего с нетривиальной НУСС ($\exists_{i,j,k}\ c_i(k) \neq c_j(k)$) величина математического ожидания средней реакции больше оптимальной

$$E\left\{\frac{1}{K}\sum_{k=1}^{K} x(k)\,|\,\pi_k\right\} > Q_k(\hat{y}_k^*)$$

Доказательство

Из теоремы 1, учитывая предположение $c_i(k) \neq c_j(k)$, вытекает, что найдется такое выходное слово ВУАПС $\bar{y}^*(k)$, что

$$E\{x(k)\,|\,\bar{y}^*(k)\} > E\{x(k)\,|\,\hat{y}^*(k)\}$$

Согласно определению ВУАПС $\pi(k)>0$ и, следовательно, $P\{y^*(k) = \bar{y}^*(k)\} > 0$.

$$E\left\{\frac{1}{K}\sum_{k=1}^{K} x(k)\,|\,\pi_k\right\} = \frac{1}{K}\sum_{k=1}^{K} E\{x(k)\,|\,\pi(k)\} =$$

$$= \frac{1}{K}\sum_{k=1}^{K}\ \sum_{y^*(k)\in Y^*} P\{y^*(k)\}\cdot E\{x(k)\,|\,y^*(k)\} >$$

$$\frac{1}{K}\sum_{k=1}^{K}\ \sum_{y^*(k)\in Y^*} P\{y^*(k)\}\cdot E\{x(k)\,|\,\hat{y}^*(k)\} = \frac{1}{K}\sum_{k=1}^{K} E\{x(k)\,|\,\hat{y}^*(k)\} =$$

$$= E\left\{\frac{1}{K}\sum_{k=1}^{K} x(k)\,|\,\hat{y}_k^*\right\} = Q_k(\hat{y}_k^*)$$

Таким образом показано, что

$$E\left\{\frac{1}{k}\sum_{k=1}^{K} x(k)\,|\,\pi_k\right\} > Q_k\,(\hat{y}_k^*)$$

теорема доказана.

Из теоремы 3 вытекает, что в отличие от РУАПС, ни один ВУАПС функционирующий в НУСС не обеспечивает оптимального взаимодействия с условной средой.

ЗАКЛЮЧЕНИЕ

В статье рассмотрен метод управления, в котором условные автоматы с переменной структурой применяются в качестве алгоритмов управления процессами, для которых построена модель в виде нестационарной условной случайной среды.

На основании представленных теорем можно предполагать, что размытый УАПС использован в системе динамического управления будет проявлять большую эффективность чем вероятностный УАПС, но эта гипотеза требует еще подтверждения, особенно в виде результатов моделирования на ЭВМ.

ЛИТЕРАТУРА

1. В.Г. Лазарев, Н.Я. Паршенков: "Игровой метод динамического управления сетью связи", в сб. "Построение управляющих устройств и систем","Наука" 1974.
2. А. Красневски: "Автоматы с переменной структурой как оптимальные алгоритмы управления потоками в сети связи", II Симпозиум "Оптимизация в Проблемах Электротехники", Закопане, 1979 /по-польски/
3. А. Красневски: "Размытый автомат в системе динамического управления распределением информации в сети связи", MTA SZTAKI Tanulmányok, Budapest, 1979.

4. K.S. Narendra, M.A.L. Thathachar - "Learning Automata - A Survey", IEEE Trans. on Systems. Man and Cybernetics, July 1974.

5. K.S. Narendra, P.A. Wright, L.G. Mason - "Application of Learning Automata to Telephone Traffic Routing and Control", IEEE Trans. on Systems, Man and Cybernetics, Nov. 1977.

6. C.V. Negoita, D.A. Ralescu - "Applications of Fuzzy Sets to System Analysis", Birkhauser Verlag 1975.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

ХАРТВИГ, Рольф

О ПРЕОБРАЗОВАНИИ ПОСЛЕДОВАТЕЛЬНОЙ СТРОКИ ОПЕРАТОРОВ ПРИСВАИВАНИЯ В ПАРАЛЛЕЛЬНУЮ, С УЧЕТОМ ИНДЕКСАЦИИ

Sektion Mathematik der Karl-Marx-Universität Leipzig
DDR - 701 Leipzig , Karl-Marx-Platz

Преобразование последовательных строк операторов в строки параллельно работающих операторов имеет значение во многих отношениях, например, при желательной - в смысле технической выгодности - реализации последовательности операторов в одновременно работающих процессах. Также при нахождении оптимальных программ такое преобразование - возможно только теоретическое - может являться очень действенным вспомогательным средством.
Здесь будем указывать предположения для возможного преобразования последовательностей операторов присваивания в параллельные на уровне языка.

I. Рассмотрим любой язык программирования или язык для описания или разработки устройств, который также допускает описание параллельной обработки. Пусть в языке имеются

множество VAR переменных,
множество EXPR выражений, где $\mathrm{VAR} \subset \mathrm{EXPR}$,
множество ASS операторов присваивания
и множество SIM т.н. параллельных предложений.

Пусть язык синтаксически однозначен, т.е. для каждого элемента языка единственным образом можно найти участвующие в его построении языковые элементы. Поэтому операторы присваивания можно понимать как пары их левых и правых частей, т.е.

$$\mathrm{ASS} \subseteq \mathrm{VAR} \times \mathrm{EXPR} ,$$

и параллельные предложения (здесь ограничиваемся параллельно работающими операторами присваивания) как некоторые n-ки операторов присваивания, т.е.

$$\mathrm{SIM} \subseteq \bigcup_{n=0}^{\infty} \mathrm{ASS}^n .$$

Семантику языка можно определить из-за его однозначности обычным способом. Пусть $\underline{W}$ - множество значений (семантическая область) и $\underline{B}$ - множество всех обложений, т.е. однозначных отображений из $\underline{VAR}$ в $\underline{W}$. Выражения и параллельные предложения (среди них операторы присваивания) интерпретируются функциями

$$\underline{value} : \underline{EXPR} \times \underline{B} \rightarrowtail \underline{W} \text{ и } \underline{val} : \underline{SIM} \times \underline{B} \rightarrowtail \underline{B} .$$

Функция $\underline{value}(-,b)$ продолжает функцию $b \in \underline{B}$ на $\underline{EXPR}$, а для $v \in \underline{VAR}$ и $S = (A_1,\ldots,A_n)$, $A_i = (v_i,H_i) \in \underline{ASS}$ имеет место $\underline{val}(S,b) = b'$, причем

$$b'(v) = \begin{cases} \underline{value}(H_j,b) & , \text{ если } j = \min\{\, i \mid 1 \leqslant i \leqslant n \wedge v = v_i\} \\ b(v) & , \text{ если } \forall i(1 \leqslant i \leqslant n \longrightarrow v \neq v_i) \end{cases}$$

Последовательное применение двух параллельных предложений определяется через $\underline{val}(S_1;S_2,b) = \underline{val}(S_2,\underline{val}(S_1,b))$.
Два параллельных предложения (или в общем две части программы) S_1 и S_2 называются $\underline{эквивалентными}$ ($S_1 \sim S_2$), если имеет место $\underline{val}(S_1,b) = \underline{val}(S_2,b)$ для всех $b \in \underline{B}$.
При этих предположениях можно показать (см. [2]):

$\underline{Теорема\ I}$. Для всякой последовательно работающей последовательности $A_1;A_2;\ldots;A_n$ операторов присваивания (или параллельных предложений) существует эквивалентное параллельное предложение $S = (A'_1,A'_2,\ldots,A'_m)$:

$$A_1;A_2;\ldots;A_n \sim S .$$

Даже больше, существует простая система правил преобразования, которая может переводить два последовательно работающих параллельных предложения $S_1;S_2$ (а также любые конечные последовательности $S_1;S_2;\ldots;S_n$) в одно параллельное предложение S и которая даже полна при соответственном предположении для эквивалентности выражений.

2. Если в языке существуют переменные с индексом, то, в общем случае, теорема не имеет места. Это вытекает из двойной роли переменных с индексом. С одной стороны, они играют роль переменных (как левая часть операторов присваивания, при синтаксическом построении выражений), с другой стороны, они не ле-

жат в области определения обложений, но представляют собой выражения для вычисления собственных переменных.

Пусть SUBSV $\subset$ VAR – множество переменных с индексами.[1] Тогда называем PVAR = VAR $\setminus$ SUBSV $\neq \phi$ множеством истинных переменных. Обложения b являются теперь однозначными отображениями из PVAR в W:

$$b : \text{PVAR} \rightarrowtail \text{W}$$

Переменная с индексом характеризуется тем, что при всяком обложении она переходит в истинную переменную (или она неопределена). Пусть эта семантика переменных с индексами описана функцией

$$\text{var} : \text{SUBSV} \times \text{B} \rightarrowtail \text{PVAR} .$$

При этом надо для $v \in$ SUBSV и $b \in$ B положить

$$\text{value}(v,b) = b(\text{var}(v,b)) .$$

Через var(v,b) = v для $v \in$ PVAR и $b \in$ B функция var распространена на множество всех переменных.

Выделяем еще множество SIMPLV так называемых простых переменных, так как они часто играют особую роль. Они определяются через

$$v \in \text{SIMPLV} \longleftrightarrow v \in \text{PVAR} \wedge \forall w \forall b(w \in \text{SUBSV} \wedge b \in \text{B} \longrightarrow \text{var}(w,b) \neq v)^2 .$$

Определение семантики параллельных предложений (определение от val) обобщается следующим образом:

- в начале, функция var параллельно применяется для вычисления всех левых частей
- к полученному параллельному предложению с истинными переменными в левых частях применяется соответственным образом модифицированное определение функции val.

И так, установили приоритет операторов в параллельных предложениях слева направо, чтобы обеспечить детерминистическую обработку при возможных повторениях левых частей. Это не влияет на одновременность присваивания значений!

---

[1] Переменные с постоянными индексами не включаем в это понятие, так как они не зависят от обложений.

[2] Согласно с этим понятием, в первой части этой статьи множества VAR и SIMPLV совпадают.

Некоторые появляющиеся относительно теоремы I трудности показывает следующий пример:

$$a_3 := c + a_3;$$
$$a_1 := (a_1 - a_3) + a_k$$

При обложениях b, где $b(k) \neq 3$ и $b(1) \neq 3$, эта система выполняет то же, что и параллельное предложение

$$(a_1 := a_1 - (c + a_3) + a_k \ , \ a_3 := c + a_3) \ ,$$

при обложениях b, где $b(k) \neq 3$ и $b(1) = 3$, то же, что и оператор

$$a_3 := a_k \ ,$$

и при обложениях b, где $b(k) = 3$, то же, что и оператор

$$a_3 := c + a_3 \ .$$

Очевидно, для независимой от обложений параллелизации систем операторов присваивания, при проявлении переменных с индексами, нужно языковые средства, которые позволяют корректное формальное выполнение подстановок вместо переменных с индексами.

Под подстановкой s понимается однозначное отображение из VAR в EXPR. Пусть SUB - множество всех подстановок

$$s : \ VAR \rightarrowtail EXPR \ .$$

В случае существования выражения, полученного в результате применения подстановки s к выражению H, оно обозначается sub(H,s). Функция

$$sub : \ EXPR \times SUB \rightarrowtail EXPR$$

является, в общем случае, частичной функцией, которую считаем известной. Пусть имеет место условие корректности

$$value(sub(H,s),b) = value(H,b_s)$$ [3]

для всех $H \in EXPR$, $s \in SUB$ и $b \in B$, причем b_s есть измененное подстановкой s обложение

$$b_s(v) = \begin{cases} value(s(v),b) & , \text{ если } v \in D(s) \\ b(v) & \text{иначе.} \end{cases}$$ [4]

---

[3] В силу сделанных предположений такая функция sub (как и value) всегда существует (как эндоморфизм в EXPR, который продолжает функцию s), причем единственная. Из-за рекурсивного построения выражений, как правило, необходимо функцию sub (как и value) также определить рекурсивно.

[4] $D(s)$ - область определения от s .

При выполнении подстановок необходимо учесть то обстоятельство, что переменные с индексами являются выражениями для определения истинных переменных и, тем самым, зависят от обложений, т.е. и от переменных. Поэтому положим

$$\underline{sub}(v,s) = s(v)$$

только для истинных переменных $v \in \underline{PVAR}$. Но для переменных с индексами необходимо описать действие подстановки некоторой функцией

$$\underline{subv} : \underline{SUBSV} \times \underline{SUB} \rightarrowtail \underline{VAR}$$ [5]

С помощью этого получим

$$\underline{sub}(v,s) = s(\underline{subv}(v,s))$$

для $v \in \underline{SUBSV}$.

Через $\underline{subv}(v,s) = v$ для $v \in \underline{PVAR}$ распространяем функцию subvна все множество VAR :

$$\underline{subv} : \underline{VAR} \times \underline{SUB} \rightarrowtail \underline{VAR} .$$

Корректность функции subv, которую предполагаем, означает, что имеет место

$$\underline{var}(\underline{subv}(v,s), b) = \underline{var}(v, b_s)$$

для всех $v \in \underline{VAR}$, $s \in \underline{SUB}$ и $b \in \underline{B}$.

Следовательно, для возможности эквивалентного преобразования систем операторов присваивания в параллельные предложения решающее значение играет следующий вопрос:
существует ли для каждого параллельного предложения S такая подстановка $\bar{S}$, чтобы измененное через $\bar{S}$ обложение действовало также как параллельное предложение S, т.е.

$$b_{\bar{S}} = \underline{val}(S,b) .$$

3. Подстановка $\bar{S}$, индуцируемая параллельным предложением S, должна допустить, чисто синтаксически, т.е. независимо от обложений, описать изменение присваивания различных правых частей к одной переменной с индексом (или к истинным переменным, полученным путем ее вычисления), которое зависит от обложений. Очевидно, что это невозможно для любых языков, т.е.

---

[5] Замечания в сноске [3] имеют место также для subv (также как для var).

без конкретных требований к множеству EXPR выражений. Укажем теперь у с л о в и я, которые имеют место для большинства интересующих нас языков, делают возможно корректное определение подстановки $\bar{S}$ и, таким образом, обеспечивают верность теоремы I.

(I) Каждую переменную с индексом $v \in \underline{SUBSV}$ можно представить в виде $v = \lambda(H_1, H_2, \ldots, H_n)$, где $H_1, H_2, \ldots, H_n \in \underline{EXPR}$ ("индексные выражения") и λ - n- местная словарная функция (так называемая "массивная функция")

$$\lambda : \underline{EXPR}^n \rightarrowtail \underline{SUBSV} .$$

Для всякой словарной функции λ, которая производит переменные с индексом, существует функция

$$\sigma_\lambda : \underline{W}^n \rightarrowtail \underline{PVAR} ,$$

которая описывает выбор истинных переменных ("компонентов массива") в зависимости от значений индексных выражений. Для этих функций имеет место:

$$\lambda \neq \lambda' \longrightarrow Im(\sigma_\lambda) \cap Im(\sigma_{\lambda'}) = \phi \quad [6] .$$

Функция $\underline{var} : \underline{SUBSV} \times \underline{B} \rightarrowtail \underline{PVAR}$ определяется через

$$\underline{var}(\lambda(H_1, \ldots, H_n), b) = \sigma_\lambda(\underline{value}(H_1, b), \ldots, \underline{value}(H_n, b))$$

и функция $\underline{subv} : \underline{SUBSV} \times \underline{SUB} \rightarrowtail \underline{VAR}$, аналогично, через

$$\underline{subv}(\lambda(H_1, \ldots, H_n), s) = \lambda(\underline{sub}(H_1, s), \ldots, \underline{sub}(H_n, s)).$$

(2) В языке существует множество COND условных выражений, где $\underline{COND} \subset \underline{EXPR}$, причем каждое $C \in \underline{COND}$ единственным образом представимо в виде

$$C = \beta_\lambda(v, w, H, H') \quad [7]$$

где λ есть словарная функция, производящая переменные с индексом, $v, w \in Im(\lambda) \cup Im(\sigma_\lambda)$, $H, H' \in \underline{EXPR}$ и β_λ есть сопоставленная функции λ словарная функция:

$$\beta_\lambda : (Im(\lambda) \cup Im(\sigma_\lambda))^2 \times \underline{EXPR}^2 \longmapsto \underline{COND} .$$

Обратно, для каждой функции λ существует словарная функ-

---

[6] $Im(\lambda)$ - область значений от λ

[7] У некоторых, подходящим образом определенных, функций σ_λ нужны только такие условные выражения, которые сами не имеют переменные как составляющие, а только индексные выражения. Сравни [I].

ция β_λ, производящая условные выражения.
Ее действие определяется через

$$\underline{value}(\beta_\lambda(v,w,H,H'),b) = \begin{cases} \underline{value}(H,b), \text{если } \underline{var}(v,b)=\underline{var}(w,b) \\ \underline{value}(H!b), \text{если } \underline{var}(v,b)\neq\underline{var}(w,b) \end{cases}$$

для всех $b \in \underline{B}$.

Замечание. Например, в языке АЛГОЛ-68 условные выражения с отношением - тождества как условие могут играть эту роль, так чтобы $\beta_\lambda(v,w,H,H')$ могло иметь вид

$$\underline{if}\ v :=: w\ \underline{then}\ H\ \underline{else}\ H'\ \underline{fi}$$

- независимо от λ, причем v,w являются переменными с индексом (т.е. так называемыми вырезками в языке АЛГОЛ-68).

При этих условиях легко можно при заданном параллельном предложении S каждой переменной v сопоставить выражение $\overline{S}(v)$, как правило условное, для которого имеет место

$$\underline{value}(S(v), b) = \underline{val}(S, b)(v) .$$

Для массивных функций λ определим множество

$$\underline{arr}(\lambda) = \{ v \mid v \in \underline{SUBSV} \wedge Im(\lambda) \vee v \in \underline{PVAR} \wedge Im(\sigma_\lambda) \}$$

и назовем его массивом, принадлежащим λ. Массив $\underline{arr}(\lambda)$ однозначно определен уже одним произвольным элементом в силу условия единственности, которое указывается в (I).
Пусть $S \in \underline{SIM}$, причем $S = (A_1, A_2, \ldots, A_n)$, $A_i \in \underline{ASS}$ и $A_i = (v_i, H_i)$. Тогда для любых $v \in \underline{VAR}$

$$\underline{ass}_S(v) = \begin{cases} H_j, \text{ если } j = \min\{ i \mid 1 \leqslant i \leqslant n \wedge v_i = v \} \\ v, \text{ если } \forall i (1 \leqslant i \leqslant n \longrightarrow v_i \neq v) \end{cases}$$

обозначает выражение, придаваемое переменной v через S.
Пусть далее для $v \in \underline{SIMPLV}$

$$\overline{S}(v) = \underline{ass}_S(v) ,$$

а для $v \in \underline{arr}(\lambda)$ рекурсивно определим

$$\overline{S}(v) = \begin{cases} \beta_\lambda(v, v_j, H_j, \overline{S'}(v)), \text{ если } j = \min\{ i \mid 1 \leqslant i \leqslant n \wedge v_i \in \underline{arr}(\lambda) \} \\ v, \text{ если } \forall i (1 \leqslant i \leqslant n \longrightarrow v_i \notin \underline{arr}(\lambda)), \end{cases}$$

причем S' есть то параллельное предложение, которое получается устранением оператора A_j из S.
Тогда имеет место

Лемма I. Для всех $S \in \underline{SIM}$ и $b \in \underline{B}$ имеет место

$$\underline{\mathrm{val}}(S,\ b) = b_{\overline{S}}\ .$$

В общем, вышеупомянутое определение от $\overline{S}(v)$ дает условные выражения ненужной сложности, потому что переменная v сравнивается со всеми переменными из того же самого массива, которые выступают как левые части параллельного предложения S. Часть этих сравнений можно познать ненужной уже синтаксическими средствами. Это относится к несколько раз появляющимся переменным и с учетом особой роли истинных переменных. Можно получить существенно более простые выражения, определение которых будет сложнее.

Пусть

$$\underline{\mathrm{firstsubsv}}_S(\lambda) = \begin{cases} v_j\ , & \text{если } j = \min\{i \mid 1\leqslant i\leqslant n \wedge \\ & \wedge\ v_i \in \underline{\mathrm{arr}}(\lambda) \wedge \underline{\mathrm{SUBSV}}\} \\ \text{не определено, если} & \forall i(1\leqslant i\leqslant n \longrightarrow \\ & v_i \notin \underline{\mathrm{arr}}(\lambda) \wedge \underline{\mathrm{SUBSV}}) \end{cases}$$

называется <u>первой переменной с индексом</u> массива $\underline{\mathrm{arr}}(\lambda)$ в S,

$$\underline{\mathrm{begv}}_S(\lambda) = \begin{cases} v_j\ , & \text{если } j = \min\{i \mid 1\leqslant i\leqslant n \wedge v_i \in \underline{\mathrm{arr}}(\lambda)\} \\ \text{не определено, если} & \forall i(1\leqslant i\leqslant n \longrightarrow v_i \notin \underline{\mathrm{arr}}(\lambda)) \end{cases}$$

называется <u>начальной переменной</u> массива $\underline{\mathrm{arr}}(\lambda)$ в S, и

$$\mathrm{BEG}_S(\lambda) = \begin{cases} \{v_j\}, & \text{если } v_j = \underline{\mathrm{begv}}_S(\lambda) \in \underline{\mathrm{SUBSV}} \\ \{v \mid v \in \underline{\mathrm{arr}}(\lambda) \wedge \underline{\mathrm{PVAR}} \wedge \exists i(1\leqslant i<j\leqslant n \wedge v=v_i)\}\ , & \\ \quad \text{если } \underline{\mathrm{begv}}_S(\lambda) \in \underline{\mathrm{PVAR}} \wedge v_j = \underline{\mathrm{firstsubsv}}_S(\lambda) & \\ \{v \mid v \in \underline{\mathrm{arr}}(\lambda) \wedge \underline{\mathrm{PVAR}} \wedge \exists i(1\leqslant i\leqslant n \wedge v=v_i)\}\ , & \\ \quad \text{если } \underline{\mathrm{begv}}_S(\lambda) \in \underline{\mathrm{PVAR}} \wedge \underline{\mathrm{firstsubsv}}_S(\lambda) - \text{не опр.} & \\ \quad , \text{если } \underline{\mathrm{begv}}_S(\lambda) - \text{не опр.} & \end{cases}$$

называется <u>началом</u> массива $\underline{\mathrm{arr}}(\lambda)$ в S.

В случае существования начальной переменной $\underline{\mathrm{begv}}_S(\lambda)$ $\underline{\mathrm{red}}(S,\lambda)$ обозначает параллельное предложение, полученное из S устранением всех операторов, левая часть которых является начальной переменной $\underline{\mathrm{begv}}_S(\lambda)$ массива $\underline{\mathrm{arr}}(\lambda)$ в S.

С помощью этих понятий можно различать все возможные случаи (см. ниже) и, кроме того, исключить все сравнения, которые можно узнать ненужными уже синтаксическими средствами.

Займемся теперь рекурсивным определением <u>выбирающего выражения</u> $\underline{\mathrm{case}}_S(v)$, сопоставленного переменной v через S.

Различаем следующие случаи (при этом пусть $w = \underline{begv}_S(\lambda)$):

(1) $v \in \underline{SIMPLV}$

(2) $v \in \underline{arr}(\lambda) \wedge (v \in \underline{BEG}_S(\lambda) \vee (v \in \underline{PVAR} \wedge w \in \underline{PVAR}) \vee \vee \underline{BEG}_S(\lambda) = \phi)$

(3) $v \in \underline{arr}(\lambda) \wedge v \notin \underline{BEG}_S(\lambda) \wedge v \in \underline{PVAR} \wedge w \in \underline{PVAR}$

(4) $v \in \underline{arr}(\lambda) \wedge v \notin \underline{BEG}_S(\lambda) \wedge (v \in \underline{SUBSV} \vee w \in \underline{SUBSV})$

Тогда пусть

$$\underline{case}_S(v) = \begin{cases} \underline{ass}_S(v) & \text{для (1) и (2)} \\ \underline{case}_{S'}(v) & \text{для (3)} \\ \beta_\lambda(v, w, \underline{ass}_S(w), \underline{case}_{S'}(v)) & \text{для (4)}, \end{cases}$$

где $S' = \underline{red}(S, \lambda)$.

Функция

$$\underline{case}_S : \underline{VAR} \longmapsto \underline{EXPR}$$

– также подстановка. С учетом всех случаев индуктивно можно доказать следующую лемму.

Лемма 2. Для всех $S \in \underline{SIM}$, $v \in \underline{VAR}$ и $b \in \underline{B}$ имеет место

$$\underline{value}(\underline{case}_S(v), b) = \underline{value}(\overline{S}(v), b) .$$

Определим теперь подстановку параллельного предложения S_1 в параллельное предложение S_2 как применение индуцированной через S_1 подстановки $\underline{case}_{S_1}$ к левым частям (с помощью функции $\underline{subv}$) и к правым частям (с помощью $\underline{sub}$) параллельного предложения S_2 с "прицепкой" предложения S_1 к результату. Эквивалентность двух параллельных предложений S_1 и S_2 оказывается равнозначным с эквивалентностью выбирающих выражений $\underline{case}_{S_1}(v)$ и $\underline{case}_{S_2}(v)$ для подходящих представителей v массивов всех выступающих как левые части параллельных предложений S_1 или S_2 переменных.

В силу леммы 1 и леммы 2, наконец, имеет место

Теорема 2. Для каждой конечной последовательности параллельных предложений

$$S_1; S_2; \ldots; S_n$$

существует эквивалентное параллельное предложение S :

$$S_1; S_2; \ldots; S_n \sim S .$$

Доказательство конструктивное. Параллельное предложение S получается в результате повторной подстановки одного параллельного предложения в следующее ("правило подстановки").

Пример. Для приведенного вначале примера с помощью правила подстановки получим эквивалентное параллельное предложение

$$(a_1 := (\underline{\text{if}}\ 1 = 3\ \underline{\text{then}}\ c + a_3\ \underline{\text{else}}\ a_1\ \underline{\text{fi}} - (c + a_3)) + \underline{\text{if}}\ k = 3\ \underline{\text{then}}\ c + a_3\ \underline{\text{else}}\ a_k\ \underline{\text{fi}}\ ,\ a_3 := c + a_3\ ) .$$

Если добавим эквивалентное преобразование выбирающих выражений ("правило эквивалентности выражений"), как второе правило преобразования, то и здесь имеем полную систему правил преобразования, потому что имеет место

Теорема 3. Если две конечные последовательности параллельных предложений эквивалентны,

$$S_1;S_2;\ \cdot\ \cdot\ \cdot\ ;S_n \sim S'_1;S'_2;\ \cdot\ \cdot\ \cdot\ ;S'_m\ ,$$

то можно преобразовать одну в другую с помощью правила подстановки и правила эквивалентности выражений.

Эта теорема подчеркивает значение нахождения синтаксических правил для эквивалентного преобразования условных выражений. Подробные определения правил преобразования, доказательства теорем и примеры содержаются в [I].

ЛИТЕРАТУРА

I Hartwig, R., Über sprachliche Ausdrucksmittel zur syntaktischen Beherrschung der Äquivalenz von Folgen simultaner Wertzuweisungen an indizierte Variablen, Preprint der Sektion Mathematik der Karl-Marx-Universität, Leipzig 1979

2 Rohleder, H., Über Umformungen von Anweisungsfolgen, Elektronische Informationsverarbeitung und Kybernetik 15(79)7, 377-382

Й. МЕТЦ

К ОПИСАНИЮ И РЕАЛИЗАЦИИ УПРАВЛЯЮЩИХ АЛГОРИТМОВ

ГДР, Технический Университет Дрезден, Секция математики

Управляющие алгоритмы могут быть описаны формальным языком, который базирует в основном на списке обработаемых параллельно событый. Для реализации описание переведется в программу универсального процессора.

Моделью управляющей системы являются множество событий E, множество переменных V, множество значений W и функции

$$f: E \to \underline{P}(V), \quad g_e: W^{f(e)} \to W^{f(e)}, \quad h_e: f(e) \times W^{f(e)} \to E$$

для всех $e \in E$. $\underline{P}(V)$ называется множеством частных множеств от V. Состояние системы в каждом моменте времени дано функцией $s: V \to W$, которая сопоставляет значение каждой переменной, и $t: E \to \underline{P}(V)$, которая сопоставляет частное множество переменных каждому событию. Сначала событие $e_o \in E$ имеет все переменные, значит $t(e_o) = V$. Если в каком - нибудь моменте времени $f(e) \subseteq t(e)$, значит что все нужные переменные события даны, то событие $e \in E$ обработано, значит что переменные из $f(e)$ меняются по g_e и переходят к другим событиям по h_e. Значения переменных тоже могут непосредственно меняться входными сигналами или являться выходными сигналами. Процесс останавливается, если нем никакого события может быть обработаемым.

Следующим образом можно конструировать соответствующая сеть Петри к этому модели. Пусть $\underline{E}$ множество событий сети Петри, $\underline{C}$ множество условий, $\underline{Pre}(\underline{e})$ множество предварительных и $\underline{Post}(\underline{e})$ множество последовательных условий для $\underline{e} \in \underline{E}$. То

$$\underline{E} = \{(e, s_e) \mid e \in E \wedge s_e \in W^{f(e)}\}$$

$$\underline{C} = \{(e, s_v, v) \mid e \in E \wedge s_v \in W \wedge v \in V\}$$

$$\underline{Pre}(e, s_e) = \{(e, s_e(v), v) \mid v \in f(e)\}$$

$$\underline{Post}(e, s_e) = \{(e', s'_e(v), v) \mid e' = h_e(v, s_e) \wedge s'_e = g_e(s_e) \wedge v \in f(e)\}$$

На этом модели базирует описующий язык для управляющих алгоритмов. Синтактическая структура записивается в форме Вакус - Наур:

<программа> ::= begin <список событий> end
<событие> ::= <список меток> : <список операторов> ;
<список меток>
<оператор> ::= <переменная> := <выражение> |
if <условие> then <список меток> |
input <список переменных> |
output <список переменных>

Списки событий и операторов образуются разъединяющим знаком ; , списки меток и переменных посредством запятой. Метки соответствуют предоставлению переменных для определенных событий. Каждая метка появляется только один раз в начале описания события.

Корректность любой программы легко проверяется, если известно соответствие между метками и переменными. В особенности, можно проверять, существуют ли в описании событий только такие переменные, которые предоставлены начальными метками, и передается ли каждая переменная какого-нибудь события при любой интерпретации одному и только одному событию. Высказывания о достижимости событий можно получить изучением недетерминированного автомата с множеством состояний $M = E^V$ и с переходной функцией $d: M \to \underline{P}(M)$, причем для всех $m \in M$

$$d(m) = \{ m' \mid m' \in M \wedge \exists e \in E: \exists s_e \in W^{f(e)}: \forall v \in V: (v \in f(e) \wedge m(v)=e \wedge m'(v)=h_e(v,s_e)) \vee (v \notin f(e) \wedge m(v)=m'(v)) \}$$

Существенные практические выгоды получаются путем объединения переменных в группы переменных.

При перевода описания управляющих алгоритмов на машинный язык процессора можно применить разные принципы. Разработана модель компилятора, которая на мелкий ЭВМ KRS 4200 вычисляет программу для микропроцессоры и моделирующую программу для KRS 4200 . При этом не каждая метка получает

элемент памяти, а каждое событие получает счетчик, который считает вступающие группы переменных. Условия, соответствующие событиям, не все время проверяются, а существует список, который принимает те события, которым только что передана группа переменных. Таком образом, число тестов уменьшалось.

Литература:

- Lehmann, T., Rechnergestützte Programmierung von Mikrorechnern für Steueraufgaben, Diplomarbeit TU Dresden, Sektion Mathematik 1979
- Metz, J., Zur Realisierung ereignisorientierter Steuerungen, Akad.d.Wiss.d.DDR, Berlin, ZKI-Informationen 1/1979

ЯН КОЛЕНИЧКА

ДВУХАДРЕСНЫЙ МИКРОПРОГРАМНЫЙ АВТОМАТ С ПОСТОЯННОЙ ПАМЯТЬЮ

Кафедра математики Педагогического факультета
г. Банска Быстрица, ЧССР

ВВЕДЕНИЕ

В настоящей модной волне микропроцессоров как-то забывается, что для целово ряда простейших управляющих устройств можно пользоваться простыми - это значит с большим экономическим эффектом - структурами микропрограмных управляющих автоматов. В настоящей статье мы хотим сказать несколько слов о концепции такого автомата, который удобен для управления такими устройствами, которым не надо работать с большой рабочей частотой (напр. программатор в автоматической стиральной машине, управление лифтом, семафорами, станками итп.) Эта ориентация позволяет создать простую "двухадресную" структуру микропрограмного автомата, у которого есть возможность получить изменение его поведения только за счет смены постоянной памяти, и вмешательство в остальную часть структуры автомата в сущности не нужно.

1. ОСНОВНАЯ КОНЦЕПЦИЯ

Место и роль микропрограмного управляющего автомата в системе показанна на рис.1. Если предположим, что в одном такте разрешается осуществить в управляемом устройстве только одну микрооперацию Y_i, то роль управляющего микропрограмного автомата заключается в том, что он должен передать в этом такте единичное значение сигнала на провод, обозначенный символом Y_i.

Информация о том, какая операция должна быть в управляемом устройстве осуществлена, закодирована при помощи переменных $X_{k+1}, X_{k+2}, \ldots, X_R$, которые представляют собой код операции.

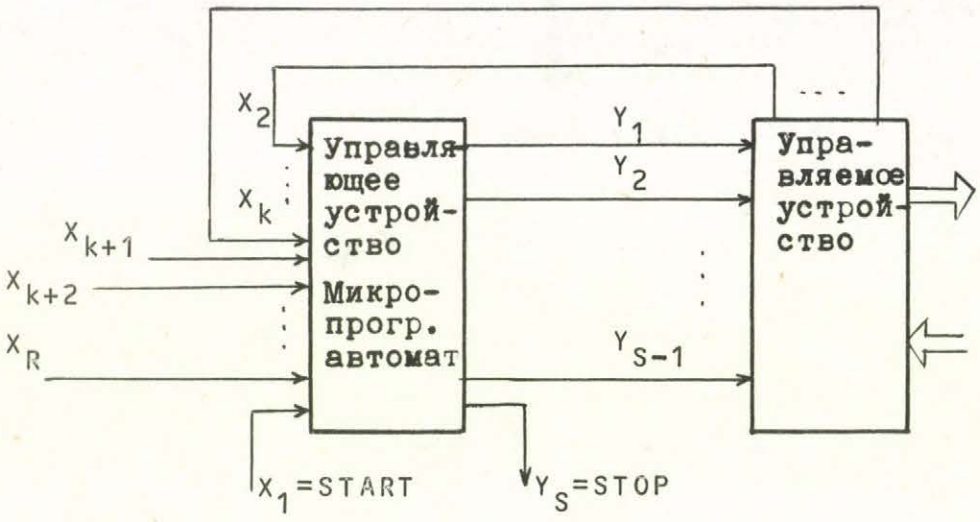


Рис.1 Место и роль микропрограмного автомата

Известно, что код операции нельзя в течение одного такта менять. (Мы можем отметоть, что код операции может быть записан внутри управляющего устройства - в его постояанной памяти).

Переменные $X_2, X_3, \ldots, X_k$ несут информацию о состоянии управляемого устройства, или иначе говоря, о ходе операции в этом устройстве. Переменная X_1 предназначена к началу действия автомата; то же и для перемнной Y_S : если $Y_S = 1$, операция окончена. Переменные X_1 до X_R мы будем называть условными переменными.

Следовательно, микропрограмный автомат для требуемой операции (которая заказана кодом операции, т.е. состояанием переменных X_{k+1} до X_R), должен передать такую последовательность сигналов (переменных) Y_i, чтобы в управляемом устройстве осуществилась именно требуемая операция.

Как известно, в концепции Вон Ноймана действие системы управляется при помощи команд записанных в виде программы в памяти. Часть команды - т.е. код операции - содержит в себе информацию о том, что надо в тот же промежуток времени сделать (т.е. какая именно операция должна быть осуществлена); другая, адресная часть команды содержит в себе информацию - эксплицитно или имплицитно выраженную - о том, какая команда и при каких ус-

ловиях будет осуществлена в следующий момент. Если мы введем другую программу (т.е. если мы изменим содержание памяти), то система будет действовать иначе, по этой новой программе.

Все это похоже на действие микропрограмного автомата и из того вытекает, что приведенную концепцию можно аналогично применить и для синтеза микропрограмного автомата, который должен управлять ходом данного множества операций, микропрограммы которых (соединенные в одну) мы можем вложить в постоянную память.

Перечень розличных подходов к синтезу микропрограмных автоматов по этой философии можно найти в [1,2]. Большинство из них мы можем характеризовать таким образом, как на рис.2.

Автомат со структурой по рис.2 содержит регистер адреса с принадлежащим декодером, блок образования адреса и постоянную память типа 2Д. На каждой строке этой памяти записана одна микрокоманда, которая состоит из двух частей: на первых S местах (слева) записана микрооперация - точнее ее код, и на остальных n местах информация, которая в виде сигналов $a_1, a_2, \ldots, a_n$ приходит в блок образования адреса, задачей которово является
- в зависимости от состояния условных переменных $x_1, x_2, \ldots, x_R$
- создать новый адрес микрокоманды, которая будет реализована в следующем такте.

Задача синтеза такого автомата заключается в основном в создании блока образования адреса. Отдельные модификации таких автоматов отличаются друг от друга именно в тем, каким образом создан блок образования адреса, т.е. каким способом создается адрес следующей микрокоманды.

Приведенная концепция микропрограмного автомата является совершенно простой и дает возможность изменить значение любого числа условных переменных из множества $\{x_1, x_2, \ldots, x_R\}$. Это очень удобно тогда, когда мы хотим удовлетворить требование высокой рабочей частоты автомата, как например в ЦВМ. Все это идет за счет большого объема постоянной памяти и сложности блока обра-

зования адреса, который, кроме того, является используемым только для одинственной микропрограммы, записанной в постоянной памяти. Если бы мы хотели добиться другого поведения автомата только сменой блока постоянной памяти (или ее содержания) мы должны были бы создать также новый блок образования адреса.

Во многих практических задачах при управлении простейшими у-

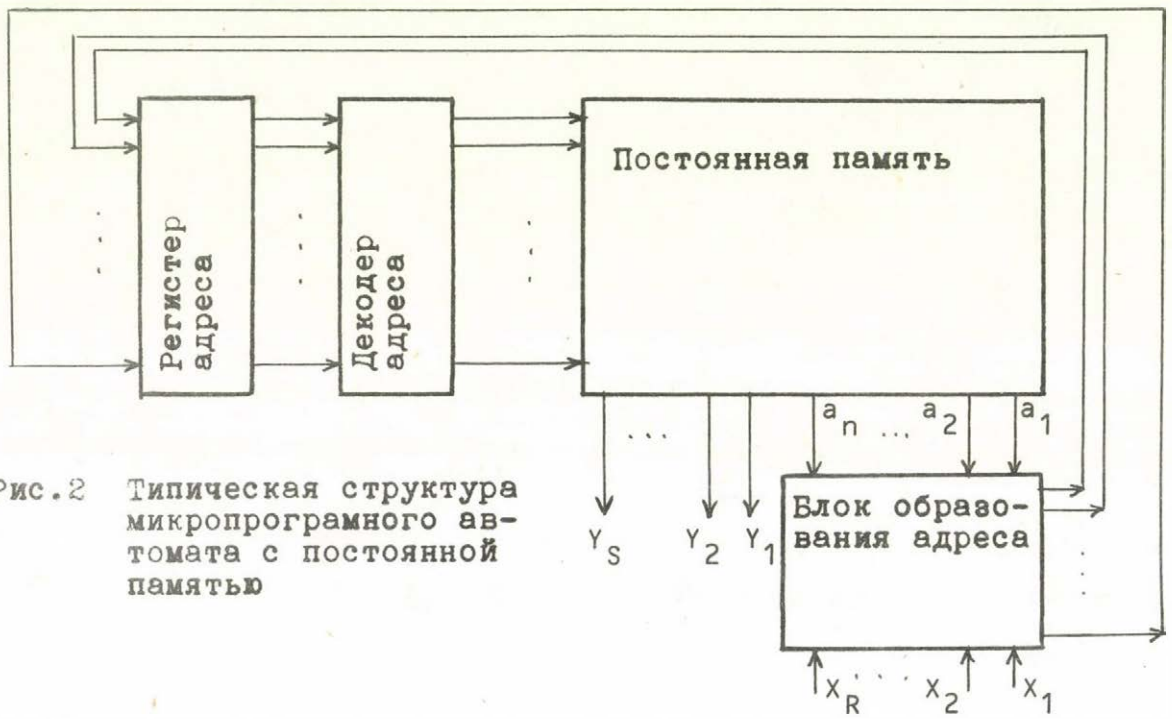


Рис.2 Типическая структура микропрограмного автомата с постоянной памятью

стройствами, у которых нет требований большой рабочей частоты, мы можем пользоваться простейшей концепцией микропрограмного автомата, которая позволяет уменьшить размер постоянной памяти и у которой изменение поведения мы получим только сменой блока постоянной памяти.

На рис.3 приведена блок-схема такого автомата, структура которого совпадает с приведенными простейшими требованиями. Эта концепция основана на следующих предположениях:

1. В каждом ориентированном пути в графе соответствующей микропрограммы между операционной вершиной Y_i и множеством ей принадлежащих операционных вершин существует только одна или никакая условная вершина.

2. В каждое ориентированное ребро, соединяющее две непосредственно связанные условные вершины x_r и x_s в графу микропрограммы (r может совпадать с s), включена одна "пустая" операционная вершина y_0, которая представляет собой так называемую "пустую микрооперацию".

3. В одном и том же самом такте T_k может быть реализована только одна микрооперация $y_i \in \{y_0, y_1, \ldots, y_S\}$, т.е. может быть генерирован только один из сигналов $y_0, y_1, \ldots, y_S$.

4. В одном и том же самом такте T_k, одна и только одна условная переменная $x_j \in \{x_1, x_2, \ldots, x_R\}$ может принимать значение единицы.

5. Каждому ориентированному ребру, соединяющему две непосредетвенно связанные операционные вершины y_r, y_s $(r \neq s)$, принадлежит условная переменная x_0 (единечное значение которой представляет собой переход без условия).

Выполнение этих предположений и требований приводит нас к тому, что к любой операционной вершине y_i графа микропрограммы принадлежат не более чем две операционные вершины y_j, y_k $(j \neq k)$, к которым можно перейти посредством не более чем двух ориентированных путей, каждый из которых содержит одну или никакую условную вершину.

Из этого вытекает, что структура автомата может быть совсем простой, потому что в каждую микрокоманду можно потом прямо записать адреса обеих микроопераций, к которым есть возможность перейти в следующем такте (т.е. после окончания текущей микрокоманды).

Из этого следует, что целый блок образования адреса можно представить в виде переключателя одного из двух возможных адресов, который управляется при помощи схемы (блока) коинциденций.

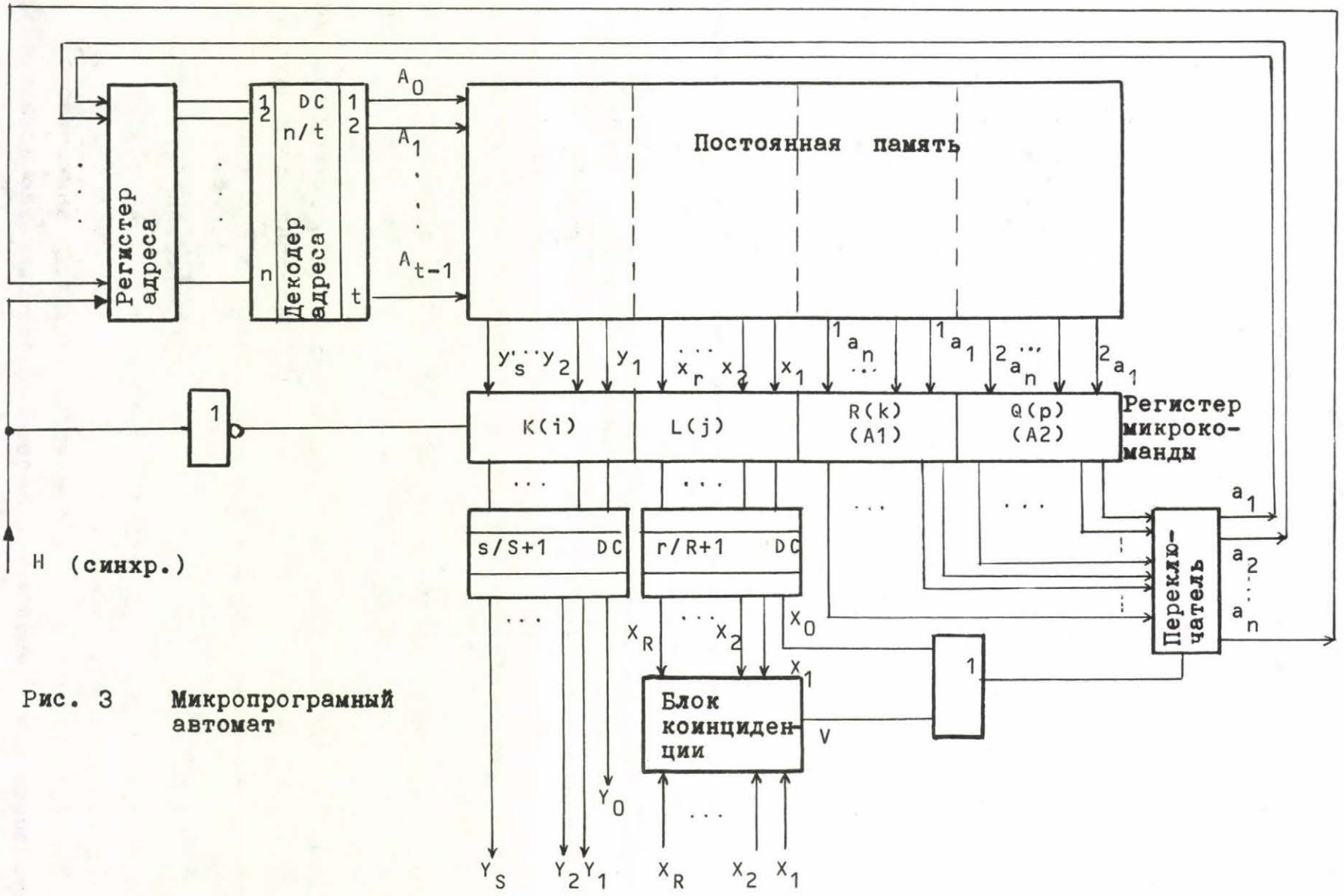


Рис. 3 Микропрограмный автомат

Определение:

Микрокомандой является упорядоченная четверка

$$< Y_i, X_j, A1_k, A2_p > ,$$

где $Y_i \in \{Y_0, Y_1, \ldots, Y_S\}$ $\quad i \in \{0,1,\ldots,S\}$

$X_j \in \{X_0, X_1, \ldots, X_R\}$ $\quad j \in \{0,1,\ldots,R\}$

$A1_k, A2_p \in \{A_0, A_1, \ldots, A_{t-1}\}$; $k,p \in \{0,1,2,\ldots,t-1\}$.

Если элементы этих множеств мы будем булевым способом кодировать так, что

Y_i кодировано состоянием $K(i)$ переменных $y_s,\ldots,y_2,y_1$

X_j кодировано состоянием $L(j)$ переменных $x_r,\ldots,x_2,x_1$

$A1_k$ (отн. $A2_p$) кодировано состоянием $R(k)$ (отн. $Q(p)$) переменных

$^1a_n,\ldots,^1a_2,^1a_1$; (отн. $^2a_n,\ldots,^2a_2,^2a_1$),

тогда упорядоченная четверка

$$< K(i), L(j), R(k), Q(p) >$$

является кодом микрокоманды в микропрограмном автомате.

2. СИНТЕЗ АВТОМАТА

Метод синтеза автомата мы покажем на примере. Пусть задан (как обыкновенно) граф микропрограммы на рис.4. Операционные вершины, проведенные черточками, пока не будем обдумивать.

Граф в этом виде не удовлетворяет всем требованиям, указанным выше, потому что существует непосредственная связь между двумя условными вершинами №2 и 3 и петля у условной вершины № 1. В эти ребра нада поэтому вложить пустую микрооперацию Y_0, т.е. вложить операционные вершины № 5 и 7 (нарысованые черточками). Дальнейшее оформление здесь уже не нужно. Поетому мы можем перейти к синтезу.

1. Каждой операционной вершине в графе принадлежит одно слово постоянной памяти (одна микрокоманда), или иначе, одна строка

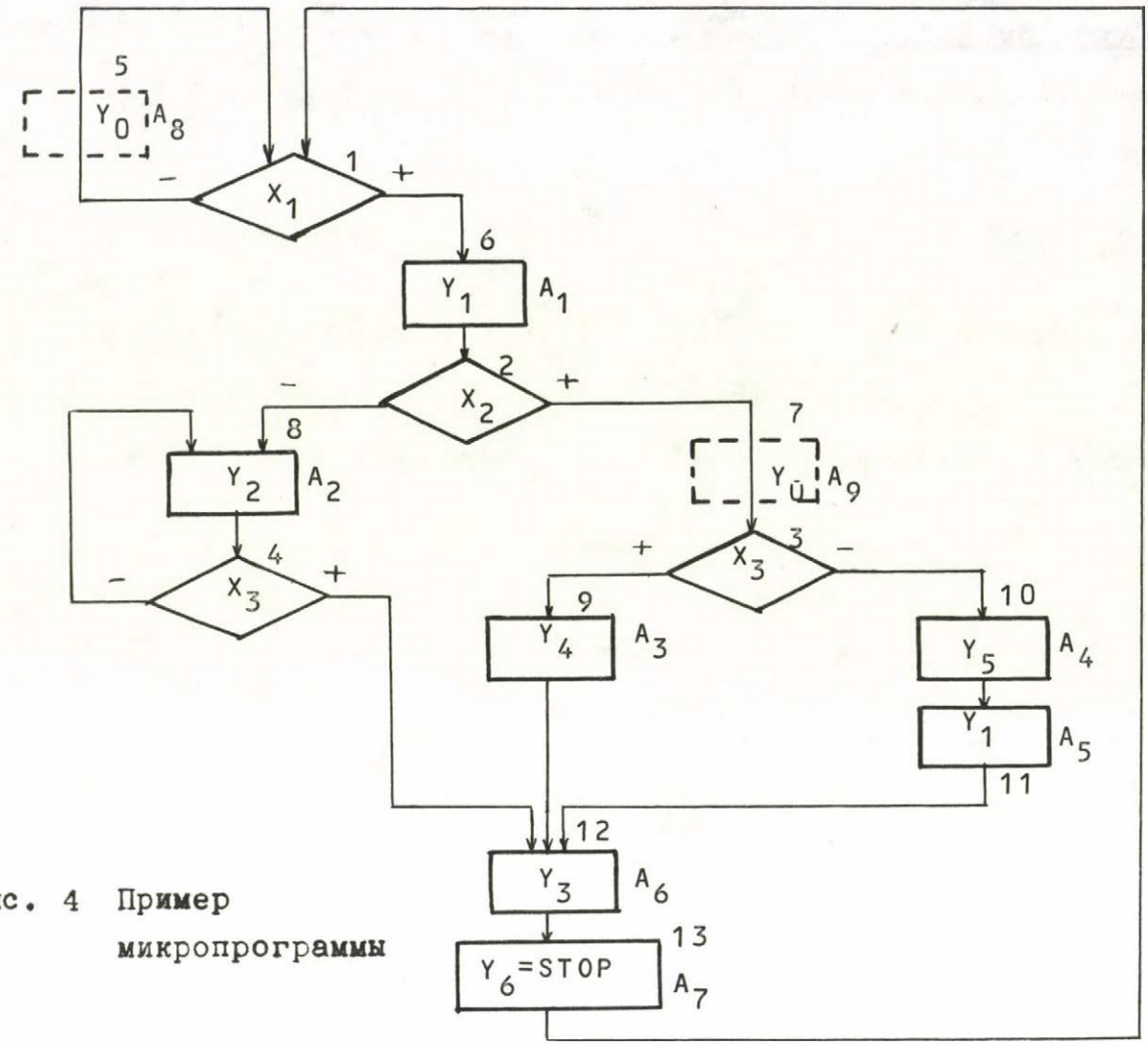


Рис. 4 Пример микропрограммы

таблицы, которой мы можем выразить содержание памяти. Поэтому к каждой операционной вершине графа мы пристроим в любой очереди один адрес A_k, и запишем это в первые два столбца таблицы 1.

2. Третий столбец таблицы нам не трудно записать при помощи графа: если после операционной вершины следует условная вершина, то в соответствующей строке в третьем столбце мы запишем условие, которое записано в условной вершине графа. Если после операционной вершины следует тоже операционная вершина, в третий столбец мы запишем механически символ X_0.

3. Содержание последних двух столбцов таблицы мы можем тоже легко записать при помощи графа. Если условие выполнено,то в

Адрес	Выход (микрооп.)	Условие	Новый адрес +	Новый адрес -
A_1	Y_1	X_2	A_9	A_2
A_2	Y_2	X_3	A_6	A_2
A_3	Y_4	X_0	A_6	-
A_4	Y_5	X_0	A_5	-
A_5	Y_1	X_0	A_6	-
A_6	Y_3	X_0	A_7	-
A_7	Y_6	X_1	A_1	A_8
A_8	Y_0	X_1	A_1	A_8
A_9	X_3	A_3	A_3	A_4

Табл.1

столбец + мы запишем тот адрес, к которому идет в графе ребро со знаком +, и в последний столбец тот адрес, к которому идет в графе ребро со знаком -. При неусловных переходах не надо определять символ в столбце таблицы - (иначе говоря,он может быть любым).

Часть Табл. 1 в толстой рамке, представляет собой в каком-то виде содержание постоянной памяти и мы можем перейти к кодированию.

4. Все символи Табл. 1 можно кодировать любым (но булевым) способом. Наиболее частым является кодирование при помощи натурального двоичного кода: в нашем случае например такое:

	$Y_3Y_2Y_1$		$Y_3Y_2Y_1$		$X_3X_2X_1$
Y_0=	0 0 0	Y_4=	1 0 0	X_1=	0 0 1
Y_1=	0 0 1	Y_5=	1 0 1	X_2=	0 1 0
Y_2=	0 1 0	Y_6=	1 1 0	X_3=	0 1 1
Y_3=	0 1 1			X_0=	0 0 0

После кодирования мы можем Табл. 1 переписать в Табл. 2.Часть в толстой рамки Табл.2 представляет собой уже прямо содержание постоянной памяти синтезируемого автомата.

5. Табл.2 содержит всю информацию о розмере постоянной памяти

автомата и о длинах регистров и декодеров автомата, и нам осталось только эту информацию применить к структуре этого автомата по рис.3. Таким образом синтез окончен.

A	K(i)	L(j)	R(k)	Q(p)
0000	0001	0010	1001	0010
0001	0010	0011	0110	0010
0010	0100	0000	0110	-
0011	0101	0000	0101	-
0100	0001	0000	0110	-
0101	0011	0000	0111	-
0110	0110	0001	0001	1000
0111	0000	0001	0001	1000
1000	0000	0011	0011	0100

Табл. 2

ЗАКЛЮЧЕНИЕ

Идеи этого синтеза мы переработали в алгоритмическую форму в виде программ KOL 1 и KOL 2 на языке BASIC для ЦВМ PDP-11. Программа KOL 1 приведет в порядок исходный граф микропрограмм в смысле выше показанных требований (граф вводится в машину в виде матрицы инциденций). Программа KOL 2 создает содержание постоянной памяти автомата в виде Табл.2, с длинами требуемых регистров.

Л И Т Е Р А Т У Р А

[1] Лазарев,В.Г.;Пийль,Е.И.: Синтез управляющих автоматов. "Энергия", Москва 1970

[2] Баранов,С.И.: Синтез микропрограмных автоматов. "Энергия", Ленинград, 1974

[3] Коленичка,Я.: Алгоритмический синтез микропрограмного автомата с постоянной памятью. Доцентская работа, 1978

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

K. SAPIECHA, K. WALCZAK, M. NOWICKI
Institute of Computer Science
Warsaw Technical University
POLAND

TEST SET GENERATION

In this paper critical comparison of the methods for generating test set to detect all the stuck-at-faults in a switching circuit is given. On this basis the necessary features of test set generation system are determined. In the following a test set generation uses in the test-generation system worked out in the Institute of Computer Science WTU is presented. For the illustration some practical examples are given.

1. INTRODUCTION

The problem how to derive a test set that will detect all the stuck-at type faults in both combinational and sequential circuits has been one of the most studied and publicated problem in the recent years. The great number of methods was proposed and evaluated. In these methods numerous assumptions are introduced which limit, more or less, their generality. First, only permanent stuck-at faults are assumed. Then a circuit irredundancy is frequently supposed, and at last the attempts for combinational and sequential circuits are usually detached.

On the basis of critical comparison of the Well-known test set generation methods a strategy will be presented which is simple, effective and very useful from practical point of view.

2. COMPARISON OF THE TEST-SET-GENERATION METHODS

In the Table 1 the classification of the well-known test set generation methods is given.

Table I

	Combinational circuits		Sequential circuits
Single faults	irredundant	redundant	
	Friedrich 1974,4 Reddy 1977,5 Schertz 1972,6		Breuer 1971,2
Multiple faults	Friedrich 1974,4 Zukow 1976,9 Thayse 1972,7 Chicoix 1977,3	Bossen 1971,1 Yau 1975,8	Zukow 1976,10

To consider the set of all possible single faults that can occur in a network one would yield complex procedure. Therefore it is important to reduce the set of faults to its subset occuring at checkpoints.

Checkpoints have the property that all single faults in an irredundant combinational network are detected iff all single faults on the checkpoints are detected. Friedrich and Davis [1974,4] assumed that the checkpoints of a combinational network are:

1) All fan-out free inputs,
2) All fanouted branches.

In S. Reddy's paper [1977, 5] it is shown that this set should be extended. The primary input nodes which fan out, have to be included in order to make the derivation of test sets for detecting single stuck-at faults valid.

Let us notice that to derive test sets to detect all single and multiple stuck-at-faults, the earlier definition of checkpoints will still be sufficient.

In the paper of Schertz and Metze [1972, 6] a new representation for faults in combinational circuits is given. The faults which are indistinguishable are identified and combined into the classes of equivalence. Moreover, the authors show that test set which detects every single fault in a circuit without "nonrestricted connected set" also detects every multiple fault in it.

For sequential circuits the discussed problem is considerably difficult. In case of sequential circuit we usually begin with a testing sequence given by designer or with a random sequence and then we generate discard tests. In Breuer's paper [1971, 2] it is shown that the random procedure appears to be effective in rapid generation of an initial test sequence which detects a large percentage of the faults. The system described in [2] will automatically switch from the random procedure to the algorithm procedure.

The problem how to derive multiple faults detection set has been the subject of a number of investigations. Unfortunately, no satisfactory solutions are available.

Bossen and Hong [1971, 1] presented the general algorithm for solving the above problem. The algorithm is established while analysing the cause-effect relationships in circuits using the minimal number of checkpoints. However, this general algorithm is not sufficient for the redundant circuits because each fault - detection set generation is usually far from optimal.

Yau and Yang gave an improvement of the method introduced in [1]. Their algorithm provides a nearly minimal test set for any combinational circuits. The minimization of test set is reached by minimization of cover of a set of representative fault functions, which has much smaller number of elements than the number of all possible distinct fault functions.

In the work of Friedrich [1974, 4] it is shown how to derive a minimal multiple-faults detection set for irredundant combinational circuit. A procedure for generating a minimal test set for single faults is performed. Then fault masking is studied and the multiple faults undetected by test set stated is derived.

This method requires the construction of masking graph and is very laborious. The different approach to the above problem is presented in the paper of Zukow [1975, 9]. After obtaining test set for single faults several procedures are utilized to derive multiple faults which are not detected by test set stated. In contradistinction to the methods previously described, these procedures are based on the gate level description of

of circuit (as a connection table of the gates). However, this method requires much time necessary to create and analyse graph of single compensations.

In the paper [10] the generalization of this method for sequential circuits is given.

The methods of generation test set for irredundant combinational circuits in simple but not always minimal way are given in the works by Thayse and by Chicoix. Thayse's method requires the generation of pair of tests for every checkpoint. In the method of Chicoix a single fault set T_s is first derived and then, by changing the value of each sensitized literal all new tests are obtained. These new tests are added to T_s in order to get a multiple-faults detection set T_m.

3. A TEST SET GENERATION METHOD

The test set generation method which is applied in the test generation system worked out in Institute of Computer Science WTU is based on the utilization of both test generator and test simulator.

The strategy leads to deriving test set detecting all single faults in any combinational sequential circuits. For this purpose fault schedul is created in which all single faults are coded. The collaboration of generator and simulator is arranged in the following manner:

1). Take from the fault schedul a fault so far undetected.
2). Generate the test for this fault.
3). Verify credibility of the test with the aid of the simulator.
4). Strike off the fault schedul faults detected by given test.
5). Repeat the points 1-4 until the fault schedul will be empty.

For combinational circuit this algorithm can be simplified by generation of the test only for checkpoints and then verification, if all faults in the circuit are covered by the generated tests. For the redundant

circuits this verification is necessary because in this case it is not always true that the test set detecting all single faults on checkpoints, detects all the faults in the while circuit as well.

4. CONCLUSION

The problem how to derive a test set that detects all the faults in any switching circuit has no general and satisfactory solution.

The proposed strategy of the application of both test generator and test simulator has the following adventages:

- the credibility of the tests is assured,
- the way of tes-set-generation for both combinational and sequential circuits is identical,
- the received test set is nearly minimal because two concepts are composed: checkpoints and test simulation.

In table 2 the comparison of the strategy considered along with the random test set generation is given.

Table 2

	random test-generation		the strategy described	
	test-sequence lenght/% of faults tested (average)	time (s)	test-sequence lenght (if 100%)	time (s)
SN7442	40/93	4,9	16	3,4
83	36/96	13,5	16	9,5
85	44/68	15,7	35	59,5
141	16/90	3,7	13	4,1
148	100/87	14,7	27	9,8
151	48/82	6,0	40	17,0
153	44/98	4,6	33	7,7
154	60/73	12,7	33	11,6
156	24/87	2,0	11	2,1
180	40/94	5,6	10	4,9
182	36/81	3,4	18	4,9
average	44/86	8,6	23/100	12,0

The strategy provides the test sets estimately twice shorter than random test sequences. The percentage of tested faults is 100 contrary to 86 (average) in case of random generation. The cost is only 50% higher.

REFERENCES

[1] Bossen D.C., Hong Y.: Cause-effect analysis for multiple fault detection in combinational networks. IEEE Trans. Comput. November 1971.

[2] Breuer M.A.: A random and an algorithmic technique for fault detection test generation for sequential circuits. IEEE Trans. Comput., November 1971.

[3] Chicoix C., Floutier D.: Fault diagonis in combinational net-works with a bidimensional boolean representation. Digital Processes, 3, 1977.

[4] Friedrich M., Davis W.A.: Minimal faults tests for combinational networks. IEEE Trans. Comput., August 1974.

[5] Reddy S.M.: Comments on "Minimal fault tests for combinational networks". IEEE Trans. Comput., March 1977.

[6] Schertz D.R., Metze G.: A new representation for faults in combinational digital circuits. IEEE Trans. Comput., August 1972.

[7] Thayse A.: Multiple-fault detection in large logical networks Philips Res. Repts. 27, 1972.

[8] Yau S.S., Yang S.: Multiple fault detection for combinational logic circuits. IEEE Trans. Comput., March 1975.

[9] Żukow M.W.: Mietod prowierki kratnych nieisprawnosti w diskrietnych ustrojstwach. Awtomatika i tielimechanika, No.7, 1976.

[10] Żukov M.W.: Mietod prowierki kratnych nieisprawnosti w diskrietnych ustrojstwach. Awtomatika i tielimechanika, No. 9, 1976.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

РАСШИРЕНИЕ РЕКУРСИВНОГО МЕТОДА ДЛЯ СОВМЕСТНОГО УПРОЩЕНИЯ СИСТЕМЫ ЧАСТИЧНО ОПРЕДЕЛЕННЫХ БУЛЕВЫХ ФУНКЦИЙ

Pásztorné Varga Katalin

Исследовательский институт вычислительной техники
и автоматизации ВАН, Будапешт

ВВЕДЕНИЕ

По теме в шестидесятых годах были достигнуты значительные теоретические результаты, затем следовалась разработка алгоритмов для ЭВМ [1],[2],[3],[4],[5] .

В семидесятых годах снова начались исследования по теме. Причиной их были:

- Плохая эффективность алгоритмов в случае систем функций, состоящих из многих функций со многими переменными, из-за большой затраты машинного времени и требуемого объема памяти.
- Возможность реализации функций, заданных в ДНФ на программируемых логических матрицах /ПЛМ/.

При разработке методов основным требованием являлось их применимость для частично-определенных функций [7],[1] .

В статье будет представлен рекурсивный метод, считающийся оригинальным среди новых методов. Этот метод будет расширен для систем функций так, чтобы было достаточно задаться упрощаемыми функциями системы функций, в любой дизюнктивной нормальной форме; следовательно, нет необходимости вырабатывать их совершенно дизюнктивную нормальную форму.

1. ОСНОВНЫЕ ПОНЯТИЯ, ОБОЗНАЧЕНИЯ И ОПРЕДЕЛЕНИЯ

Пусть будет $f(x_1, x_2, \ldots, x_n)$ частично-определенная функция. Функции f_1, f_o, f_ϕ определим следующим образом:

$$f_1(\underline{\alpha}) = \begin{cases} 1 \text{ если } f(\underline{\alpha}) = 1 \\ 0 \text{ иначе} \end{cases}$$

$$f_o(\underline{\alpha}) = \begin{cases} 1 \text{ если } f(\underline{\alpha}) = 0 \\ 0 \text{ иначе} \end{cases}$$

$$f_\phi(\underline{\alpha}) = \begin{cases} 1 \text{ если } f(\underline{\alpha}) = \phi \text{ /неопределена/} \\ 0 \text{ иначе} \end{cases}$$

где: $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, и

$\alpha_i = 0, 1$

Мы говорим, что $\underline{\alpha} \in B^n$ является

"точкой 1" если $f(\underline{\alpha}) = 1$

"точкой 0" если $f(\underline{\alpha}) = 0$

"точкой ϕ" если $f(\underline{\alpha}) = \phi$ /неопределенности/.

$\hat{f} = f_1 \vee f_\phi$ называется верхней границей функции f.

$\underset{\wedge}{f} = f_1$ называется нижней границей функции f.

Функция $f_c = f_1 \vee \gamma f_\phi$ /или $f_c = f_1 \vee \gamma \bar{f}_o$/ называется совместной с функцией f .

2. РЕКУРСИВНЫЙ МЕТОД

Пусть

$$f_{x_{i=\alpha}} = f(x_1, x_2, \ldots, x_{i-1}, \alpha, x_{i+1}, \ldots, x_n) \qquad (1)$$

Для полностью определенной функции f всегда справедливо:

$$f = x_i \cdot f_{x_i=1} \vee \bar{x}_i \cdot f_{x_i=0} \qquad (2)$$

Очевидно, что множество "точек 1" производной булевой функции f всегда разделимо на три подмножества по переменной x функции f:

- точки, где значение координаты x равно 1, но соседняя по x "точка 0" ($x_{i_1} \cdot B_{i_1}$);
- точки, где значение координаты x равно 0, но соседняя по x "точка 0" ($\bar{x}_{i_1} \cdot C_{i_1}$);
- точки, соседние по x (A_{i_1}) .

Пусть оператор S_i означает запись функции f по ее переменной x_i в виде (3)

$$S_{i_1}(f) = f = \underbrace{f_{x_{i_1}=1} \cdot f_{x_{i_1}=0}}_{A_{i_1}} \vee x_{i_1} \cdot \underbrace{f_{x_{i_1}=1} \cdot \bar{f}_{x_{i_1}=0}}_{B_{i_1}} \vee \bar{x}_{i_1} \cdot \underbrace{f_{x_{i_1}=0} \cdot \bar{f}_{x_{i_1}=1}}_{C_{i_1}} \qquad (3)$$

где подфункции A_{i_1}, B_{i_1}, C_{i_1} , покрывающие соседние пары точек {α, β} по x_{i_1}.

Применение S к (3) по x_j означает применение S к подфункциям A_{i_1}, B_{i_1}, C_{i_2}. В случае, если $j = i_2$, получим

$$S_{i_2}(S_{i_1}(f)) = S_{i_2}(A_{i_1}) \vee x_{i_1} \cdot S_{i_2}(B_{i_1}) \vee \bar{x}_{i_1} \cdot S_{i_2}(C_{i_1}) \qquad (4)$$

Очевидно, что применение S к f по различным переменным возможно до тех пор, пока каждая функция в (4) будет равна 0 либо 1.

Обозначаем элементарные конъюнкции через:

$p^{(x_i)}$ - если x_i

$p^{(\bar{x}_i)}$ - если $\bar{x}_i$

} входит в конъюнкцию

$p^{(-)x_i}$ - если x_i^{α} не входит в конъюкцию

где $$x^{\alpha} = \begin{cases} x \text{ если } \alpha = 1 \\ \bar{x} \text{ если } \alpha = 0 \end{cases}$$

Если $q^{(x_i)}$, $q^{(\bar{x}_i)}$, $q^{(-)x_i}$ простые импликанты функции f, очевидно

$$q^{(-)x_i} \to A_i \ , \quad q^{(x_i)} \to x_i B_i, \quad q^{(\bar{x}_i)} \to \bar{x}_i C_i \ .$$

Из этого следует

Теорема 1 [8, 9, 10]: Если в (4) каждая подфункция постоянная, то после выполнения операций мы получим все простые импликанты.

Это значит, что если мы итеративно применим S к функции f, то получаем все простые импликанты.

Пусть обозначает

P(f) - дизюнкцию всех простых импликантов функции f.

Тогда из (3) можно получить следующию рекурсивную формулу:

$$P(f) = P(A_{i_1}) \vee x_{i_1}(P(B_{i_1}) \setminus P(A_{i_1})) \vee \bar{x}_{i_1}(P(C_{i_1}) \setminus P(A_{i_1})) \qquad (5)$$

это значит, что $P(A_{i_1})$ является областью неопределенности как для $P(B_{i_1})$ так и для $P(C_{i_1})$.

Очевидно, что

$P(0) = 0$

$P(1) = 1$ и $P(f) = 1$ если $\bar{f} = f_o \equiv 0$

$P(f) = f$ если $f = x$

Для иллюстрации рекурсивного метода посмотрим следующий пример.

Пример 1. Применение рекурсивного метода для полностью опре-

деленной функции.

Пусть

$$f = x_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1\bar{x}_2\bar{x}_4$$

$$\bar{f} = x_2 \vee \bar{x}_1x_4$$

Тогда

$$P(f) = P(\underbrace{\bar{x}_2x_3\bar{x}_4 \vee \bar{x}_2\bar{x}_3\bar{x}_4}_{A_1}) \vee x_1 P(\underbrace{\bar{x}_2x_3x_4 \vee \bar{x}_2\bar{x}_3x_4}_{B_1}) \vee$$

$$\vee \bar{x}_1 \underbrace{P(0)}_{C_1}$$

так как

$$\bar{A}_1 = x_2 \vee x_4 \ , \quad \bar{B}_1 = (B_1)_o = x_2$$

$$P(f) = [P(0) \vee x_2P(0) \vee \bar{x}_2P(\underbrace{x_3\bar{x}_4 \vee \bar{x}_3\bar{x}_4}_{C_2^{(A_1)}})] \vee$$

$$\vee x_1 [P(0) \vee x_2P(0) \vee \bar{x}_2P(\underbrace{x_3x_4 \vee \bar{x}_3x_4}_{C_2^{(B_1)}})]$$

так как

$\overline{C_2^{(A)}} = x_4$ и $\overline{C_2^{(B)}} = (C_2^{(B)})_o = 0$ и поэтому $P(C_2^{(B)}) = 1$

$$P(f) = [\bar{x}_2\bar{x}_4 \vee x_3P_4(0) \vee \bar{x}_3P_4(0)] \vee x_1\bar{x}_2$$

$$P(f) = \bar{x}_2\bar{x}_4 \vee x_1\bar{x}_2$$

Из такого простого примера видно, что проблема то, что много операций нужно выполнить над функциями для чего много машинного времени и много машинной памяти требуется.

Возможные облегчения в машинной реализации метода:

1/ Не всегда нужно определить все простые импликанты.

2/ Особенностями функций A, B, C можно воспользоваться для замены вычисления их значений с верификацией выполнения условий их существования.

3/ Применимы эвристические методы.

Процедура нахождения простых импликантов на базе теоремы 1. проиллюстрирована деревом /рис. 1./.

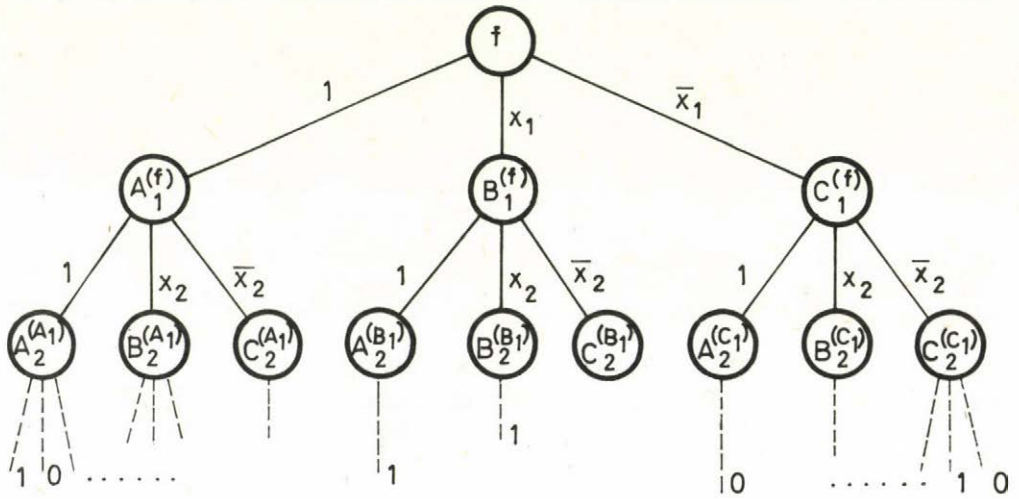


Рис. 1

На рис. 2. видно дерево для примера 1.

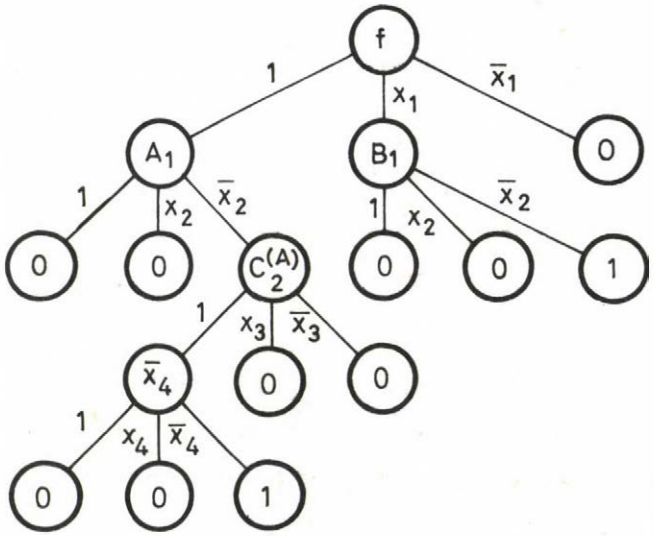


Рис. 2.

После этого можно установить, что

1/ Пути дерева оканчивающиеся на 1 определяют простые импликанты.

2/ Необходимы только пути оканчивающиеся на 1.

3/ При неизбыточном покрытии, одновременно достаточно отыскать один путь оканчивающийся на единицу. Таким образом, получаем простой импликант P и процедура начинается заново для функции $\varphi = f \cap \bar{p}$.

4/ Для частично-определенных функций рекурсивный метод не изменяется только принимаем, что $\bar{f} = f_o$.

Что касается рекурсивных методов, разработанных в литературе, можно констатировать, что [9] и [10] базируется на совершенной нормальной форме, [9] использует точки 1 и Ø, но [10] точки 1 и 0. В [8] используется произвольная дизюнктивная нормальная форма f_1 и f_o .

3. РЕАЛИЗАЦИЯ МЕТОДА

При реализации метода применяется матрица, каждая строка которой соответствует элементарной конъюнкции дизюнктивной нормальной формы функции. Столбцы матрицы соответствуют переменным функции. Элементами матрицы могут быть 0, 1 и -. В определенной строке $\alpha = (0,1)$ означает то, что соответствующая переменная является элементом конъюнкции в виде x^{α} . На рисунке 3 показана репрезентация функции

$$f = x_1\bar{x}_2x_4 \vee \bar{x}_1x_5 \vee x_2x_3\bar{x}_5 \vee x_2\bar{x}_4$$

x_1	x_2	x_3	x_4	x_5
1	0	-	1	-
0	-	-	-	1
-	1	1	-	0
-	1	-	0	-

~ $f(x_1,x_2,x_3,x_4,x_5)$

Рис. 3.

Частично определенная функция задается функциями f_1 и f_o с помощью двух матриц P и Q в форме $f = P/Q$ где

P соответствует функции f_1
Q соответствует функции f_o

Пример 2: $f = f_1 \vee \gamma f_o$ где

$$f_1 = xy\bar{z} \vee uxy \vee \bar{x}u, \qquad f_o = \bar{y}\bar{z}\bar{u} \vee \bar{x}\bar{z}\bar{u} \vee x\bar{y}$$

Тогда

$$f(x, y, z, u) = \frac{\begin{matrix} 1\ 1\ 0\ - \\ 1\ 1\ -\ 1 \\ 0\ -\ -\ 1 \\ -\ 0\ 0\ 0 \\ 0\ -\ 0\ 0 \\ 1\ 0\ -\ - \end{matrix}}{} = \frac{P}{Q}$$

Для определения подфункций в (3) или (4) и для верификации значений подфункций в этой репрезентации определим следующие операторы:

1. Оператор редукции по x_i. Обозначаем через R_{x_i}. Оператор $R_{x_i}(P/Q)$ стирает столбец переменного x_i из P/Q.

2. Пересечение по x_i^α. Оператор обозначается через $I_{x_i^\alpha}$. Оператор $I_{x_i^\alpha}(P/Q)$ стирает конъюнкции из P где x_i^α не встречается и из Q где $x_i^{\bar{\alpha}}$ встречается. Стирает столбец x_i из P/Q.

3. Покрытия по элементарной конъюнкции p. Оператор обозначается через C_p. Оператор $C_p(P/Q)$ стирает из P все конъюнкции k_i, для которых $k_i \to p$.

В следующем примере применим эти операторы на f примера 2 и покажем связь между подфункциями (3), (4) и результат применения операторов R, I, C.

Пример 3:

$$R_x(P/Q) = \begin{matrix} - & 1 & 0 & - \\ - & - & - & 1 \\ - & - & 0 & 0 \\ - & 0 & - & - \end{matrix} = \frac{(f_1)_{x=0} \quad V(f_0)_{x=1}}{(f_0)_{x=0} \quad V(f_0)_{x=1}}$$

$$I_x(P/Q) = \begin{matrix} - & 1 & 0 & - \\ - & 1 & - & 1 \\ - & 0 & - & - \\ - & 0 & 0 & 0 \end{matrix} = \frac{P_x}{Q_x}$$

$$I_{\bar{x}}(P/Q) = \begin{matrix} - & - & - & 1 \\ - & 0 & 0 & 0 \\ - & - & 0 & 0 \end{matrix} = \frac{P_{\bar{x}}}{Q_{\bar{x}}}$$

$p = xu$

$$C_p(P/Q) = \begin{matrix} 1 & 1 & 0 & 0 \\ 0 & - & - & 1 \\ - & 0 & 0 & 0 \\ 0 & - & 0 & 0 \\ 1 & 0 & - & - \end{matrix} \Big\} \; f \cap \bar{p}$$

После этого введем две теоремы важных для алгоритма, реализующего рекурсивного метода.

Теорема 2. [8] : Если $P_{x_i^\alpha} \cap Q_{x_i^{\bar{\alpha}}} = 0$ то существует простой импликант функции f содержащей x_i^α .

Теорема 3. [8] : Если $Q_{x_i^\alpha} = 0$ в $P_{x_i^\alpha} / Q_{x_i^\alpha} = I_{x_i}$, то x_i^α является простым импликантом для $x_i^\alpha P_{x_i^\alpha}$

В дальнейшем опишем алгоритм нахождения одной неизбыточной дизюнктивной формы при зафиксированном порядке переменных

Алгоритм

1. $f := f_1/f_o = P_o/Q_o$

2. $j := 0$

3. $i := 0$, $k := 1$

4. $i := i+1$

5. Получение $I_{x_i}(P_j/Q_j)$ и $I_{\bar{x}_i}(P_j/Q_j)$

6. Если $P_{jx_i} = P_{j\bar{x}_i} = 0$, то переход к п. 15.

7. Если $P_{jx_i} \cap Q_{j\bar{x}_i} \neq 0$, то $\alpha := 1$ и переход к п. 10.

8. Если $P_{j\bar{x}_i} \cap Q_{jx_i} = 0$, то переход к п. 15.

9. $\alpha := 0$

10. $p := p \cap x_i^{\alpha}$

11. Если $Q_{jx_i^{\alpha}} \neq 0$, то $k := j$, $j := 1$ $P_j/Q_j := P_{jx_i^{\alpha}}/Q_{jx_i^{\alpha}}$ и возврат к п. 4.

12. p простой импликант. $j := 0$, $P_j/Q_j := C_p(P_o/Q_o)$.

13. Если $P_o \equiv 0$, то мы получили покрытие функции. СТОП.

14. Возврат к п. 2.

15. $k := j$, $j := 1$, $P_j/Q_j := R_{x_i}(P_k/Q_k)$ переход к п. 4.

4. РАСШИРЕНИЕ РЕКУРСИВНОГО МЕТОДА ДЛЯ СИСТЕМ ФУНКЦИЙ

Обозначаем через $\mathfrak{F}$ системы функций:

$$\mathfrak{F} = \{f^{(1)}, f^{(2)}, \ldots, f^{(u)}\} \quad \text{где}$$

$$\mathfrak{F}_1 = \{f_1^{(1)}, f_1^{(2)}, \ldots, f_1^{(u)}\}$$

$$\mathfrak{F}_o = \{f_o^{(1)}, f_o^{(2)}, \ldots, f_o^{(u)}\}$$

Легко видеть, если для операторов останутся в силу теоремы 2 и 3, то при соответствующем расширении алгоритм без изменения определяет неизбыточность покрытия $\mathfrak{F}$.

Репрезентация $\mathfrak{F}$ в P/Q матрице похожно остается только каждая конъюнкция получает индекс показывающий принадлежность к компоненту функции.

Операторы мало изменяются.

1. Редукция R_{x_i} не изменяются.

2. Пересечение $I_{x_i}^{\alpha}$ не изменяются.

3. Покрытия $C_{p,ind}$ из $k_i \to p$ конъюнкции стирает только такие k_i, где индексы одинаковые. В другом случае только изменяет индекс k_i.

Пример 4. Репрезентация $\mathfrak{F}$ и вычисление операторов.

$$\mathfrak{F} = \{f^{(1)}, f^{(2)}\}$$

$$f_1^{(1)} = xy \vee \bar{x}\bar{y}\bar{z}$$

$$f_1^{(2)} = yz\bar{u} \vee \bar{x}z\bar{u} \vee \bar{x}\bar{y}\bar{z}u$$

$$f_0^{(1)} = x\bar{y}\bar{z} \vee \bar{y}zu \vee \bar{x}\bar{y}$$

$$f_0^{(2)} = x\bar{y}\bar{z}\bar{u} \vee x\bar{z}u \vee \bar{y}zu \vee \bar{x}\bar{y}\bar{z}$$

$$\mathfrak{F}(u,z,y,x) = \frac{P}{Q} = \frac{\begin{array}{cccc|cc} - & - & 1 & 1 & 0 & 1 \\ - & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & - & 1 & 0 \\ 0 & 1 & - & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{array}}{\begin{array}{cccc|cc} - & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & - & 1 & 1 \\ - & - & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & - & 1 & 1 & 0 \\ - & 0 & 1 & 0 & 1 & 0 \end{array}}$$

$$R_u(P/Q) = \begin{array}{cccc} - & - & 1 & 1 \\ - & 0 & 0 & 0 \\ - & 1 & 1 & - \\ - & 1 & - & 0 \\ - & 0 & 0 & 0 \\ \hline - & 0 & 0 & 1 \\ - & 1 & 0 & - \\ - & - & 1 & 0 \\ - & 0 & 0 & 1 \\ - & 0 & - & 1 \\ - & 0 & 1 & 0 \end{array} \quad \begin{array}{cc} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ \hline 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{array}$$

$$I_u(P/Q) = \begin{array}{cccc} - & 0 & 0 & 0 \\ \hline - & 0 & 0 & 1 \\ - & 1 & 0 & - \\ - & - & 1 & 0 \\ - & 0 & - & 1 \\ - & 0 & 1 & 0 \end{array} \quad \begin{array}{cc} 1 & 1 \\ \hline 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{array} = \frac{P_u}{Q_u}$$

$$(p,ind) = (yz\bar{u},\ 10)$$

$$C_{p,ind}(P/Q) = \begin{array}{cccc} - & - & 1 & 1 \\ - & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ \hline & & Q & \end{array} \quad \begin{array}{cc} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ \hline & \end{array}$$

В алгоритме проверяется $P_{x_i}^{\alpha} \cap Q_{x_i}^{\bar{\alpha}} = 0$ или нет. $P_{x_i}^{\alpha} \cap Q_{x_i}^{\bar{\alpha}} = 0$, если пересечение множеств конъюнкций пусто или если это пересечение не пусто, но нет общих индексов в $P_{x_i}^{\alpha}$ и в $Q_{x_i}^{\bar{\alpha}}$.

Так как теорема 2 и 3 верны и для этих операторов, поэтому алгоритмом определяется покрытие $\tilde{\sigma}$.

ЗАМЕЧАНИЕ

Машинная реализация алгоритма на ЭВМ CDC 3300 показала эффективность метода. В таблице 1 задано необходимое время выполнения алгоритма.

Число функций	Число переменных	Метод традиционный	Метод рекурсивный
3	4	0'17	0'11
3	4	0'28	0'15
3	7	1'47	0'24
3	8	3'54	0'47
3	8	1'40	0'51
1	4	0'10	0'07
1	10	3'44	1'45

Таблица 1.

Предельное значение требуемого объема памяти равно трехкратному числу конъюнкций, присутствующих в матрице P/Q.

Л И Т Е Р А Т У Р А

[1] R. McNaughton, B. Mitchell: The Minimality of Rectifier Nets with Multiple Outputs Incompletely Specified, Journal of Frankl. Inst. Vol. No. 6. 1957. 457-479.

[2] T.C. Bartee: Computer Design of Multiple-Output Logical Networks, IRE Transactions on EC. Vol. EC-10. No. 3. March. 1961. 21-30.

[3] E.J. McCluskey, Jr. and J. Shorr: Essential Multiple-Output Prime Implicants,
Microwave Res. Inst. Symp. ser. 12. 1962. 437-457.

[4] Я.А. Хетагуров, В.В. Первов, И.В. Брусенков: Один способ синтеза комбинационных логических схем на ЦВМ, Кибернетика, № 5., 1969.

[5] S.B. Akers: A diagrammatic Approach to Multiplevel Logic Synthesis, IEEE Transactions on EC. Vol. EC-14. No. 4. Apr. 1965. 171-181.

[6] L. Frécon: Écriture atomique des fonctions booléennes Automatisme - Tome XVI, No. 6-7 - juin-juillet 1971.

[7] Pásztorné, Varga Katalin: Nem teljesen meghatározott Boole-függvények szintézise $\{\wedge, \vee, \neg\}$, {NAND} ill. {NOR} bázisában, MTA Számitástechnikai Központja Közlemények, 1972. 3. szám.

[8] J.C. Torgue-K. Pásztor Varga-P. Azema: Couverture Irredondante des Fonctions Booléennes Definites par leurs Monomes Vrais et Faux, Fonctions Simultanees, Acta Cybernetica 1974. Tomes 3.

[9] P. Azema, M. Diaz, J.C. Torgue: Minimisation of Incompletely Specified Switching Functions, Electronics Letters 27. July 1972. Vol. 8. No. 15.

[10] E. Morreale: Rekursive Operators for Prime Implikant and Irredundant Normal Form Determination, IEEE Transactions on Computer Vol. C-19. No. 6. june 1970. 504-509.

Глава 2.: ДОКЛАДЫ СИМПОЗИУМА НКС 1980 ГОДА

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

ВЫЧИСЛЕНИЕ ЛОГИЧЕСКИХ ТЕСТОВ МЕТОДОМ ДВОЙНОГО СОГЛАСОВАНИЯ

Йожеф Сираи

Институт по координации вычислительной техники

1. ВВЕДЕНИЕ

В настоящей публикации рассмотрены одиночные и многократные ошибки отказа логических сетей: такие ошибки, которые выражаются в отказе отдельных точек сети на постоянном логическом уровне.

Типы учитываемых сетей: комбинационный и синхронно-последовательный. Для обоих типов предполагается, что сеть задана с эквивалентной схемой, построенной на комбинационных элементах с одним или несколькими выходами. Для упрощения изложения вопроса наличие ошибки предполагается только на выходах элементов, а также на первичных входах.

В последние годы стали известны многочисленные методы, которые пригодны для разработки последовательностей входных сигналов для выявления ошибок из-за отказов /тестов/. Алгоритм D [1] и метод булевских разностей [2], а также их разновидности одинаково пригодны для того, чтобы создать тестпрограмму для обнаружения всех распозноваемых ошибок комбинационных сетей. Первый метод работает с логическими значениями, а последний выполняет операции над логическими функциями.

Согласно обзору, приведенному в работе [3], методы, пригодные для комбинационных сетей, можно также расширить на последовательные сети. Здесь основная трудность заключается в том, что задача - из-за распространения ошибок на запоминающие устройства - даже в случае предположения наличия одиночных ошибок таит в себе расчет тестов по многократным ошибкам. Несмотря на то, что как алгоритм D , так и метод булевских разностей можно расширить на многократные ошибки [3], последнее приводит к

такому значительному увеличению объема вычислений, что сильно ограничивает практическую применяемость этих методов. По этой причине возникает серьезная потребность в таком практическом методе, который всегда пригоден для нахождения теста к распознаваемым ошибкам комбинационных сетей, причем без того, чтобы при наличии одиночных или многократных ошибок в сети в нем оказалось какое-то заметное расхождение.

В настоящей публикации представлено такое решение. Данное решение, в сущности, разработано посредством значительного улучшения и обобщения так называемой общей активизации, рассмотренной в работе [4], и оно основано на обработке логических значений, исходящих с первичных и вторичных выходов. В дальнейшем рассмотрим применение этого принципа к комбинационным и синхронно-последовательностным сетям, а затем приводим его качественную оценку, сравнивая его с алгоритмом D и методом булевских разностей.

2. ДВОЙНОЕ СОГЛАСОВАНИЕ ДЛЯ КОМБИНАЦИОННЫХ СЕТЕЙ

В настоящем разделе рассмотрим комбинационные сети, о которых предполагается, что они построены на логических элементах с одним выходом.

Пусть будет вектор первичных входных и выходных переменных какой-то комбинационной сети:

$$\bar{x} = (x_1, x_2, \ldots, x_n) \qquad \bar{z} = (z_1, z_2, \ldots, z_m)$$

При этом точки сети должны быть пронумерованы таким образом, чтобы выход какого-то элемента должен иметь порядковый номер больше, чем номер любого из его входов. Пусть будет i порядковый номер первичного входа с переменной x_i. Ошибка из-за отказа α ($\alpha = 0,1$) i-той точки сети обозначается через (α). Логическое отрицание /инвертирование/ обозначается через верхнюю запятую. И наконец: обозначение сетевого пути, включающего в себя точки i,j,...,p следующее: P(i-j-...-p).

Предполагаем, что контрольный вектор $\bar{x}_t$ может обнаружить $q \geqslant 1$ одновременно присутствующих ошибок из-за отказа на первичном выходе z_j сети, и к этому должны представить, что вектор x_t присутствует на первичных входах. Если сеть является безошибочной, то z_j показывает логическое значение β, и при этом логические значения всех точек сети согласованы между собой. Назовем это состояние состоянием нормальной согласованности.

Если теперь в сеть включим q ошибок, тогда z_j принимает значение β' /то есть это является необходимым и достаточным условием обнаружения ошибок/, и при этом возникает так называемое состояние неправильной согласованности. Это означает, что логические значения, на которых получаются отказы, не находятся в соответствии с $\bar{x}_t$ в то время, когда все остальные значения согласуются со значениями при правильной логической работе.

В соответствии с умственным экспериментом задачу формирования $\bar{x}_t$ можно также сформулировать следующим образом: Необходимо найти такую входную комбинацию, при которой состояния нормальной согласованности и неправильной согласованности отличаются друг от друга хотя бы в одном первичном выходном значении. Для достижения этой цели z_j присвоим значение β, где β произвольно 0 или 1, а потом пытаемся создать такую входную комбинацию, которая соответствует:

а/ с одной стороны, правильному значению β выхода z_j в безошибочной сети, и

б/ с другой стороны, неправильному значению β' выхода z_j в ошибочной сети.

Для решения задачи вычисления одновременно производятся в двух областях: в так называемых нормальной и ошибочной областях. Это, в свою очередь, означает, что для каждой точки учитываются два логических значения, а именно значение, вычисленное в безошибочной сети и значение, вычисленное в неисправной сети. Такая пара значений называется спаренным /двойным/ значением, где первым указывается правильное значение.

В наших вычислениях в одной области будем работать тремя логическими значениями, а именно: со значением логического "0", значением логической "1" и безразличным значением, которое обозначается через d .

Результаты основных операций /инверсии, логического И, логического ИЛИ/ по этим трем значениям представлены в таблице № 1. Семантическая интерпретация значения d: в данных фазах вычислений вместо d произвольно можно поставить значение 0 или 1 для точки, которой присвоено d без того,чтобы это привело бы к противоречию.

Множество спаренных значений можно создать и таким образом, что составляется декартово произведение от множества $\{0,1,d\}$, взятое само по себе, а это дает следующий результат:

$$\{0/0,\ 0/1,\ 0/d,\ 1/0,\ 1/1,\ 1/d,\ d/0,\ d/1,\ d/d\}.$$

Последнее означает, что вычисления производятся в логической системе с 9 значениями, где первые составляющие 9 спаренных значений попадают в область нормальных значений, а вторые значения, в свою очередь, в область ошибочных значений. В этой логической системе первая и вторая составляющие результата какой-то операции, выполненной над двумя значениями, будет результатом той же операции, выполненной над первыми и вторыми составляющими операдов.

a	a'	И			ИЛИ		
		0	1	d	0	1	d
0	1	0	0	0	0	1	d
1	0	1	0	1	d	1	1
d	d	d	0	d	d	1	d

Таблица № 1: Таблица верности логической системы с тремя значениями.

Исходными значениями будут следующие: $z_j = \beta/\beta'$, а в точках с ошибкой α они составляют d/α. К полученным таким образом исходным значениям применяется так называемая процедура согласования /**consistency operation**/, которая входит в состав алгоритма D. Согласование, выполненное спаренными значениями, называется двойным согласованием. Это необходимо выполнить таким же образом, как в алгоритме D с одиночными значениями, но со следующими ограничениями:

а/ два спаренных значения согласованы тогда и только тогда, если как их нормальная составляющая, так и ошибочная составляющая также согласованы;

б/ в области неправильных значений не требуется согласовать значения, на которых получается отказ, со значениями, предшествующими им в потоке сигналов.

Конечно, в процессе выполнения согласования необходимо проследить только за определенными логическими значениями, а составляющие d не требуется проверить. Если в процессе получается противоречие, тогда следует изменить присвоение переменного значения, выполненное последним, а затем стереть все значения, которые были присвоены после этого, и продолжать вычисление.

Количество шагов изложенного выше процесса значительно можно уменьшить по следующим двум соображениям.

Само сабой разумеется, что в тех точках сети, через которые не проходит сигнал от какой-то ошибочной точки к первичным выходам, две составляющих спаренных значений не должны отличаться. Эти точки называются пассивными точками. Со значением пассивных точек, с одной стороны, значительно уменьшается количество спаренных значений, которые в определенных случаях можно выбрать, а с другой стороны, для такой точки в дальнейшем вместо двух требуется согласовать всего лишь одно логическое значение. Здесь следует отметить, что для нас достаточно определить точки, через которые сигнал распространяется от ошибочной точки

к выходу z_j . /Такие точки называются потенциально активными точками./ Это получается таким образом потому, что все остальные точки либо являются пассивными, либо же они не участвуют в процессе вычислений, начатых для z_j . Для определения множества потенцильно активных точек достаточно проходить по сети один раз в направлении распространения сигналов, исходя при этом из ошибочных точек, и один раз в направлении, противоположном распространению сигналов, исходя из выхода z_j , причем для выполнения этого служит простой и быстро реализуемый алгоритм [4]. Здесь следует отметить, что в дальнейшем выход z_j будет учитываться только в случае, если он доступен хотя бы из одной ошибочной точки через какой-то определенный путь распространения сигналов.

Второе соображение касается начальных значений z_j. Здесь трудность заключается в том, что заранее не знаем, что именно выбрать в качестве правильного значения и что в качестве ошибочного, поэтому используется произвольный выбор. Однако в случае однократной ошибки не требуется повторно выполнить процедуру согласования с перестановленными значениями z_j , если первоначальный выбор не дал бы нужного результата. То есть, если пара начальных значений z_j в месте ошибки приводит к такой паре значений 0/1 или 1/0 /так называемому активному значению/, которая, в свою очередь, противоречит уровню отказа, тогда все присвоенные до сих пор составляющие спаренных значений можно переставить, этим снимается противоречие и можно продолжить вычисление. В месте ошибки всегда получается активное спаренное значение, так как для согласования активного спаренного значения на выходе z_j хотя бы по одному пути распространения ошибки должны присутствовать исключительно только активные значения. Если количество ошибок больше единицы, тогда процесс перестановки не всегда можно выполнить в одном и том же заходе согласования.

На основании вышеизложенных для создания теста обнаружения $q \geqslant 1$ одновременно присутствующих ошибок рекомендуется использовать следующий алгоритм:

В качестве начального значения соответствующего выхода z_j выбираем 0/1, а ошибочным точкам в области неправильных значений присвоим соответствующие значения, на которых получается отказ. После этого выполняется двойное согласование, подбирая к этому элементы сети в порядке уменьшения идентификаторов их выходов. Если $q=1$, и в последовательности активных значений по пути между ошибочной точкой i и выходом z_j получается противоречие при i, тогда следует выполнить перестановку спаренных значений для последовательно подобранных точек кроме i, а потом продолжить согласование. Если $q>1$, тогда выполнить двойное согласование для $z_j = 0/1$. Если при этом последнее будет неуспешным, тогда необходимо полностью повторить процедуру со значением $z_j = 1/0$. В случае состоятельного решения тест состоит из значений первичных входов, входящих в область нормальных значений. Если для z_j не удалось получить решения, тогда брать следующий первичный выход и выполнить для него двойное согласование.

В качестве примера рассмотрим сеть, представленную на рис. 1, где необходимо найти тест для ошибки 7(0). Соединения, соответствующие потенциально активным точкам, на рисунке проведены жирными линиями. Так как активное значение, присутствующее на входе вентиля в точке 10, противоречит уровню ошибки, необходимо выполнить перестановку. После перестановки значений алгоритм дает результат, представленный на рис. 2. Как видно из рисунка, тест $\bar{x} = (0,d,1,0,1,0)$ одновременно распространяет ошибку по пути P(7-10-11-14-16) и P(7-10-12-16), а путь P(7-9-14-16) он блокирует распространение ошибки.

В следующем примере требуется найти тест для тройной ошибки 1(1), 5(0) и 8(1) в сети, представленной на рис. 3. Согласно полученному результату тест $\bar{x} = (1, 1, 1, 0)$ ошибку 5(0) распространяет по пути P(5-7-8-9), а ошибку 8(1) по пути P(8-9) к выходу, а распространение ошибки 1(1) он блокирует.

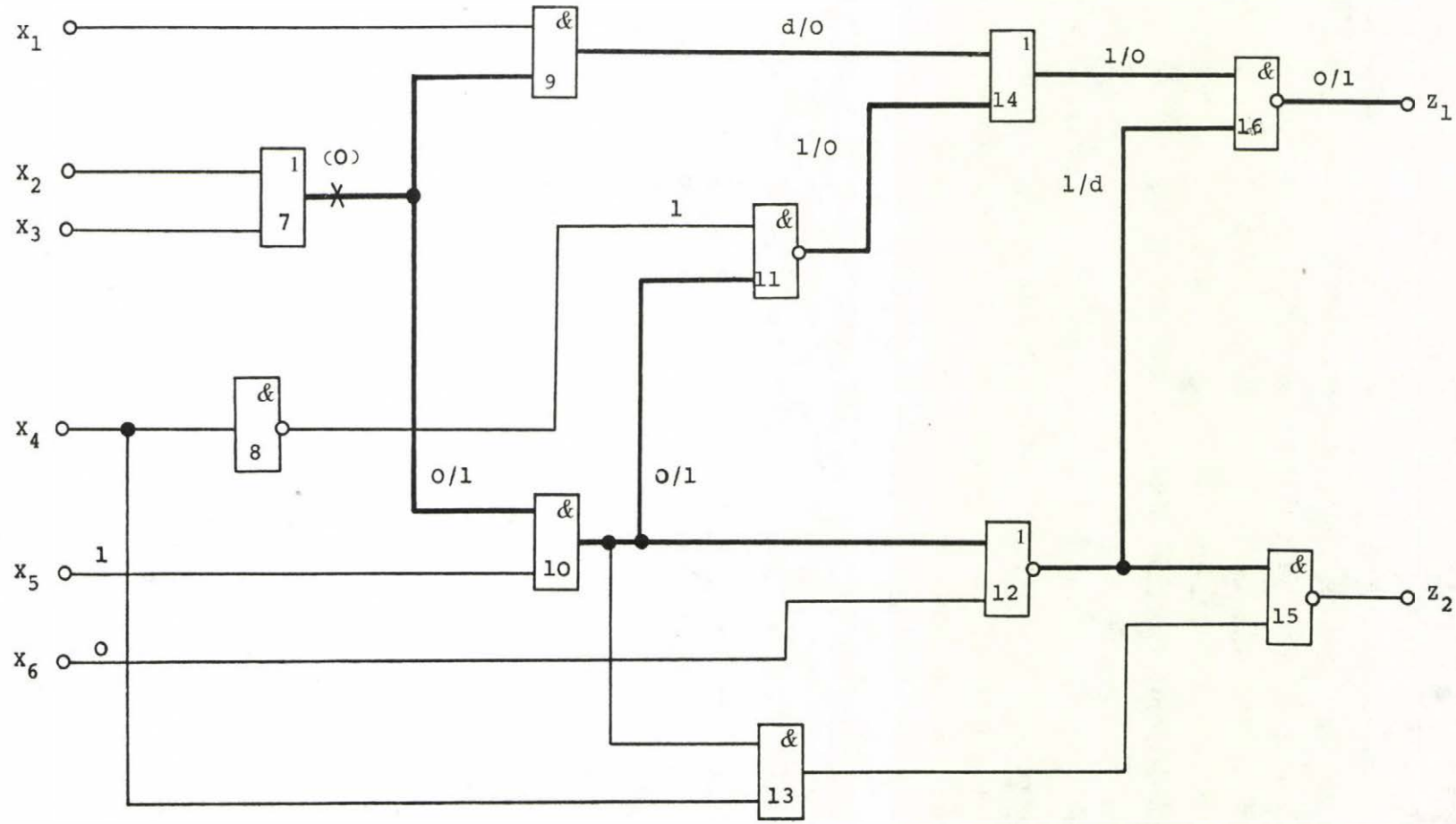


Рис. 1. Двойное согласование перед перестановкой

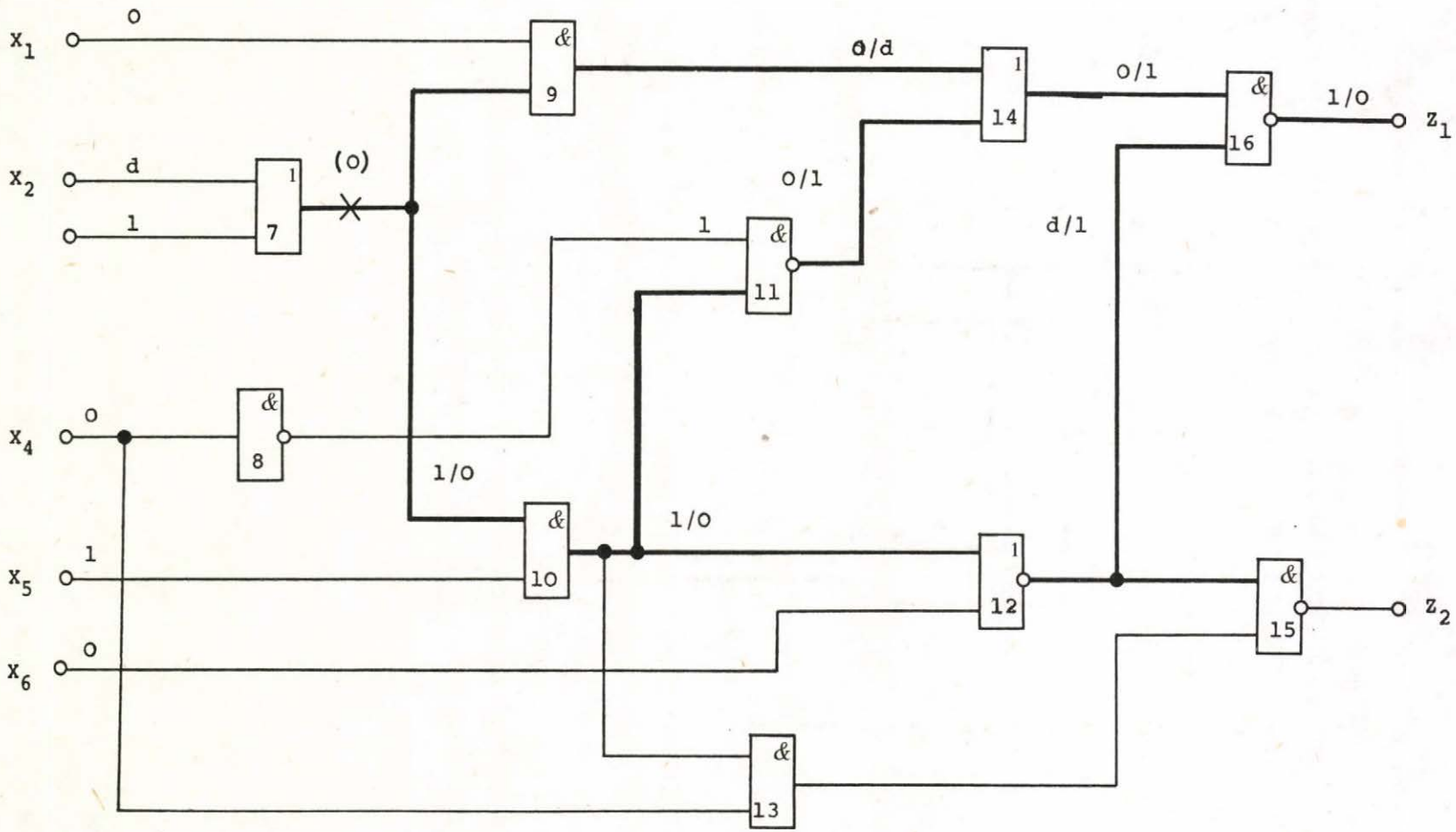


Рис. 2. Двойное согласование после перестановки

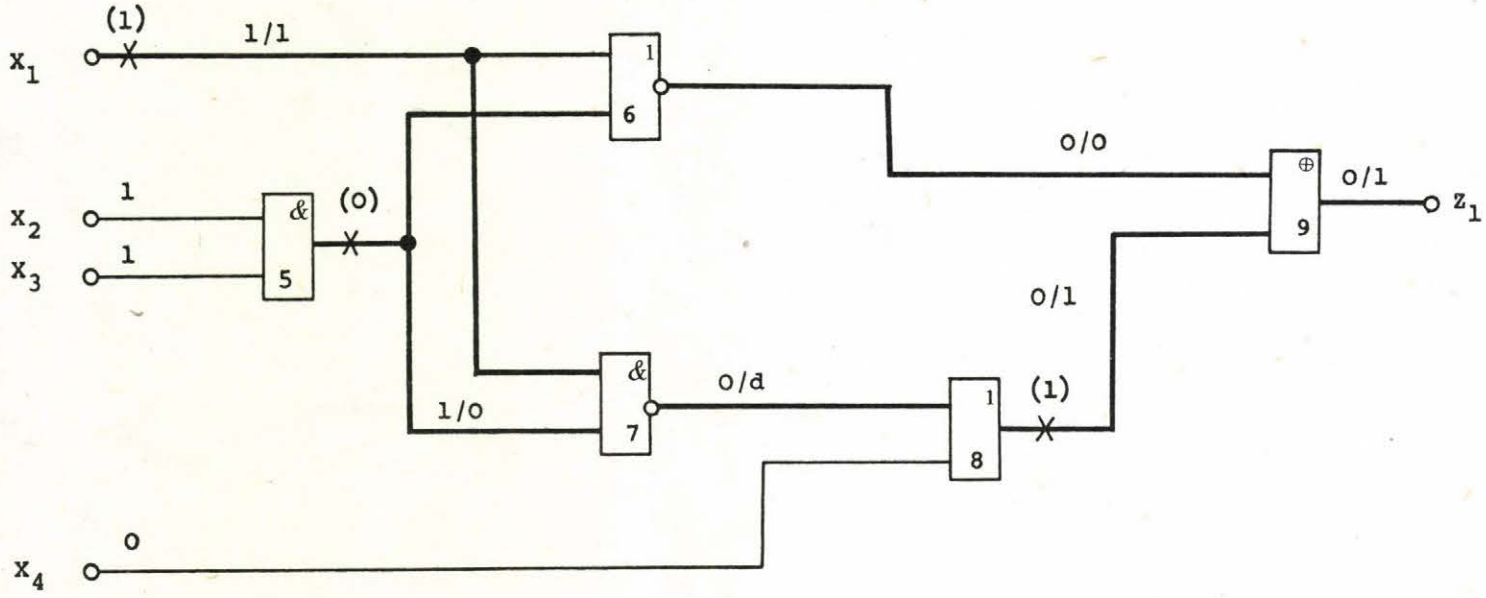


Рис. 3. Вычисление тестов для многократной ошибки

3. КОМБИНАЦИОННЫЕ СЕТИ НА УРОВНЕ МОДУЛЕЙ

В настоящем разделе принцип двойного согласования распространяется на такие комбинационные сети, которые построены на модулях БИС с несколькими выходами. Такими модулями являются, например, общеизвестные элементы ПЗУ или элементы ПЛМ. Как известно, в случае сетей на уровне вентильных схем вычисления по согласованию наиболее выгодно можно выполнить тогда, если используются простые импликанты вентилей. Процедуру легко можно распространить на такие модули, для которых известны множества простых импликантов [5]. Однако, пока нереально рассчитывать на то, что будем иметь множества простых импликантов элементов БИС, или можем решить их определение. Так как в настоящее время самым реальным предположением является, что известны множества минтермов модулей, в дальнейшем будем работать с минтермами.

Пусть будут для модуля M_i векторами входных и выходных переменных $\bar{x}^i = (x_1^i, x_2^i, \ldots, x_s^i)$ и $\bar{z}^i = (z_1^i, z_2^i, \ldots, z_w^i)$. Множество минтермов какой-то выходной функции модуля z_j^i по входным переменным модуля обозначается через E_j^i, а дополнение по множеству всех входных комбинаций модуля, E_j^i в свою очередь, обозначается через $\tilde{E}_j^i$. Очевидно, что $\tilde{E}_j^i$ не что иное, как множество минтермов обратной функции z_j^i. В дальнейшем минтерм, представленный какой-то двоичной комбинацией, будем выражать десятичным значением комбинации.

Пусть какой-то модуль M_i имеет w выходных переменных. Легко можно понять, что для модуля M_i какая-то входная комбинация $\bar{x}_0^i$ тогда и только тогда дает в качестве результата $z_j^i = 1$, если $\bar{x}_0^i \in E_j^i$, или $z_j^i = 0$, если $\bar{x}_0^i \in \tilde{E}_j^i$, где $j=1,2,\ldots,w$. Далее, если $z_j^i = d$, тогда любая входная комбинация будет согласована с этим значением. Следовательно, для того, чтобы найти входную комбинацию, согласованную с какой-то выходной комбинацией, необходимо формировать сечение соответствующих множеств минтермов выходов, имеющих значения, отличающиеся от значения

d. Например, множество всех входных комбинаций, дающих выходную комбинацию $\bar{z}^i = (0,1,d,0)$ можно вычислить следующим образом:

$$\tilde{E}_1^i \cap E_2^i \cap \tilde{E}_4^i$$

Если работа ведется со спаренными значениями, тогда необходимо выполнить одновременное согласование в области нормальных значений и области неправильных значений. Поиск соответствующей входной комбинации к выходным комбинациям обеих областей производится согласно вышеизложенным правилам. Пусть количество минтермов, приводящихся к выходной комбинации, попадающей либо в область нормальных значений модуля, либо в область его неправильных значений, будет соответственно N_n или N_f . При этом количество возможностей выбора, в результате которых получается даннаяпара выходных значений, будет $N_n \cdot N_f$, где возможны все пары элементов двух согласуемых множеств, подбирая первую составляющую какой-то пары из одного множества, а ее вторую составляющую из другого множества. В нашем случае необходимо обратить как двоичные, так и десятичные данные. После того, как выполнен выбор минтерма какого-то модуля, его необходимо преобразовать в двоичную форму для того, чтобы получить значения двоичных разрядов для отдельных точек входа. Между прочим, алгоритм двойного согласования можно выполнить таким же образом, как это представлено в предыдущем разделе настоящей публикации.

В дальнейшем представим вычисления для сети из четырех модулей, представленной на рис. 4. Модули одинаковы во всех отношениях, и они имеют следующие множества минтермов:

$$E_1^i = \{3,4,7\} \qquad E_2^i = \{0,1,3,5\}$$

где $i = 1,2,3,4$. Индексы переменных модулей возрастают по схеме включения сверху вниз. Во входной комбинации какого-то модуля входному двоичному значению x_j^i соответствует вес 2^{j-1} . Требуется определить тест для двойной ошибки $z_2^1(0)$ и $z_1^2(1)$. Дополнения для множеств минтермов при $i = 1,2,3,4$ будут следующие:

$$\tilde{E}_1^i = \{0,1,2,5,6\}, \quad \tilde{E}_2^i = \{2,4,6,7\},$$

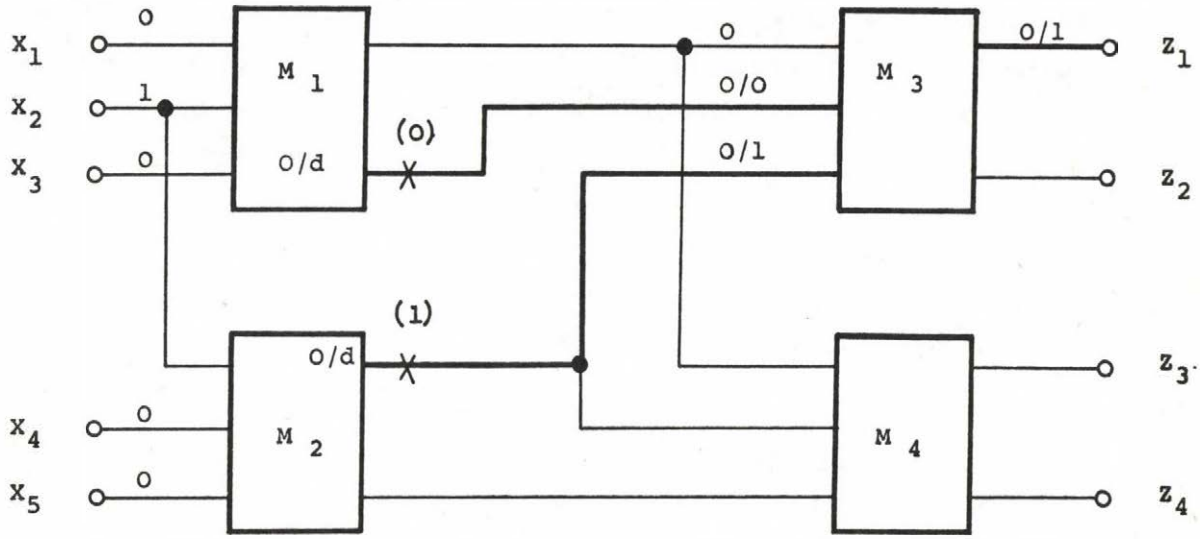


Рис. 4. Двойное согласование на уровне модулей

Двойное согласование начинается с того, что z_1 присвоим значение 0/1, при котором требуются пары минтермов из $\tilde{E}_1^3$ и E_1^3. Первой парой будет 0/3, что для точки, имеющей отказ на 0 включает в себя значение 0/1, то есть в ней содержится противоречие. Следующий выбор входной кобинации 0/4, причем здесь не будет противоречия. Этим для M_2 прослеживаемой выходной комбинацией будет $\bar{z}^2 = (0/d, d)$, а для M_1, в свою очередь, $\bar{z}^1 = (0, 0/d)$, не требуется соглавовать. Множество минтермов, присваиваемых M_2 будет:

$$\tilde{E}_1^2 = \{0, 1, 2, 5, 6\},$$

а множество минтермов, присваиваемых M_1:

$$\tilde{E}_1 \cap \tilde{E}_2 = \{2, 6\}.$$

Если для M_2 выбирается минтерм 1, а для M_1 - минтерм 2, тогда получается согласованный контрольный вектор, что имеет вид $\bar{x} = (0, 1, 0, 0, 0)$.

С точки зрения реализации на ЭВМ самыми критическими данными являются множества минтермов. Эти множества соответствуют спискам из десятичных чисел. Если для представления минтермов выбрали бы структуру списков, тогда для схем, которые встречаются на практике, получились бы слишком большие потребности в памяти и продолжительность обработки. Вместо этого выбираем двоичное представление множеств, что как с точки зрения потребности в памяти, так и времени выполнения является наиболее целесообразным решением. Такое представление удобно можно описать посредством так называемой матрицы верности. Матрицей верности модуля M_k, имеющего s входов и w выходов, является двоичная матрица

$$T_k = \left[t_{ij}^k \right]$$

которая имеет w строк и 2^s столбцов, и $t_{ij}^k = 1$ тогда и только тогда, если $j-1$ является десятичным минтермом для выходной функции z_i^k модуля; в противном случае $t_{ij}^k = 0$.

Как видно, вектор-строки матрицы T_k наиболее плотно представляют множества минтермов отдельных выходных функций. Например, в случае модуля, имеющего 10 входов и 10 выходов, для хранения матрицы верности модуля в памяти требуется место, составляющее всего лишь 10240 двоичных разрядов, то есть 1,25 Кбайтов. Что касается операций между множествами, легко можно понять, что сечения /разрезы/ получаются как логические произведения по двоичным разрядам соответствующих двоичных векторов, а дополнение означает инвертирование всех разрядов вектора. Как известно, эти операции можно реализовать простыми и быстро выполняемыми машинными командами.

4. ДВОЙНОЕ СОГЛАСОВАНИЕ ДЛЯ СИНХРОННО-ПОСЛЕДОВАТЕЛЬНЫХ СЕТЕЙ

Синхронно-последовательную сеть можно моделировать согласно рис. 5. Здесь $\bar{x}$ и $\bar{z}$ векторы первичных входных и выходных переменных соответственно, $\bar{y} = (y_1, y_2, \ldots, y_p)$ векторов вторичных входных переменных /текущее состояние/, а $\bar{y} = (y_1, y_2, \ldots, y_p)$, в свою очередь, вектор вторичных выходных перемнных /следующее

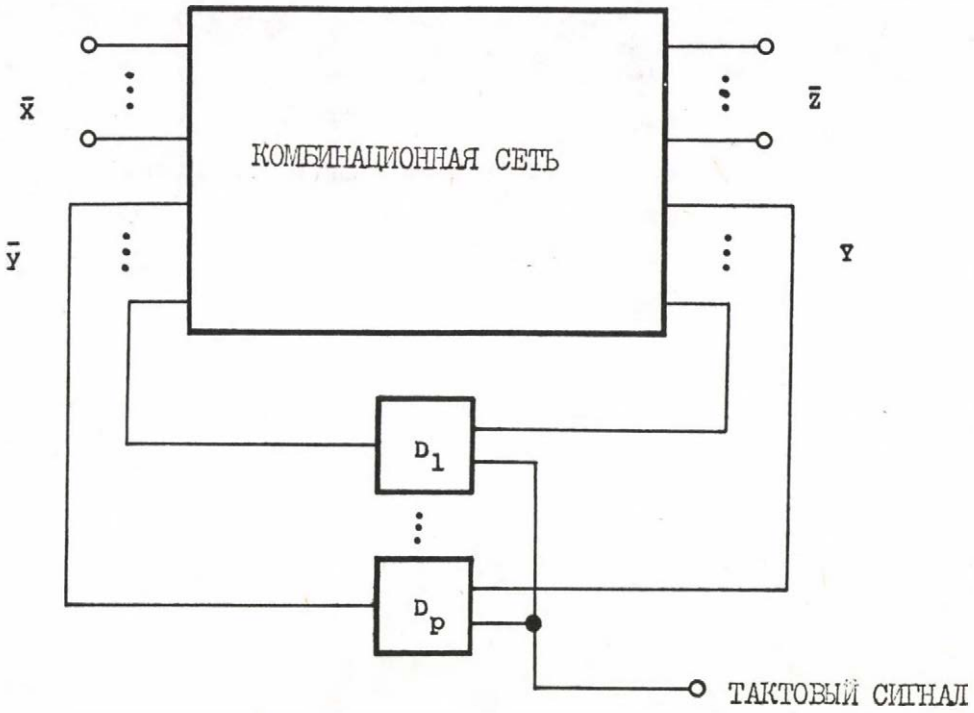


Рис. 5. Модель синхронно-последовательной сети

состояние/, а петли обратной связи включают в себя запоминающие схемы D_p, управляюемые от тактовых сигналов.

Распространение методов расчета тестов, пригодных для комбинационных сетей, на последовательные сети основано на том, что распространение ошибки необходимо выполнить для комбинационного блока модели согласно рис. 5. Обобщение соответствующих принципов приведено в работах [3] и [4]. Для выполнения расчетов над комбинационной частью сетевой модели очень выгодно можно использовать алгоритм двойного согласования, особенно в случае, если требуется заниматься распространением многократных ошибок. Последняя ситуация может получиться даже в случае одиночных ошибок: воздействие ошибки передается на выходы одного или нескольких запоминающих элементов, что на вторичных входах комбинационного блока выражается в форме виртуальной ошибки отказа.

Для того, чтобы процедуру согласования можно было использовать для последовательной модели, необходимо ввести четвертое, одиночное логическое значение, так называемое неизвестное значение, которое обозначается через u . Его интерпретация: логическое значение точки, которой присвоено u, априорно фиксированно 0 или 1, однако в самом деле действительное значение неизвестно. Таблицы верности, введенные для четырех значений, представлены в таблице № 2.

a	a'
0	1
1	0
d	d
u	u

И

	0	1	d	u
0	0	0	0	0
1	0	1	d	u
d	0	d	d	d
u	0	u	d	u

ИЛИ

0	1	d	u
0	1	d	u
1	1	1	1
d	1	d	d
u	1	d	u

Таблица № 2: Таблицы верности логической системы с четырьмя значениями.

Декартово произведение от множества {0, 1, d, u} взятое само по себе, дает множество из 16-и элементов, то есть двойное согласование для последовательных сетей выполняется в логической системе из 16-и значений: то есть одна точка сети теоретически может принимать 16 возможных спаренных значений. Однако, необходимо заметить, что применение u может потребоваться только на выходах запоминающих схем, а именно перед началом выполнения цикла согласования. При согласовании необходимо учитывать только одно, что логическое значение составляющей $\bar{y}$, которой присвоено u, кроме самого себя не противоречит только значению d, то есть для него нельзя задать ни 0, ни 1.

Если правильные/ошибочные значения вторичных входов рассчитываются посредством моделирования ошибок последовательно генерированных контрольных векторов, согласно работам [3] и [4], тогда перед началом цикла согласования спаренные значения y_i

$(i = 1,2,\ldots,p)$ следует установить следующим образом:

1. Если значение y_i неизвестно, тогда $y_i = u/u$.
2. Если известно правильное значение y_i и оно равно β, и при этом:
   а/ значение y_i безошибочно, тогда $y_i = \beta/\beta$
   б/ к y_i распространена ошибка, тогда $y_i = \beta/\beta'$
   в/ к y_i ненадежно распространена ошибка, (так называемая ошибка "star" [3]), тогда $y_i = \beta/u$

Ко всему этому необходимо еще добавить, что за исключением $y_i = \beta/\beta$ точку y_i во всех изложенных выше случаях следует считать такой точкой, откуда в ходе согласования ошибка может распространяться к выбранному первичному или вторичному выходу, то есть при этом точка y_i является потенциально активной точкой.

В качестве примера создаем контрольный вектор распространения ошибки для синхронно-последовательной сети, представленной на рис. 6. На рисунке не указаны петли обратной связи. Предполагаем, что при наличии входной последовательности $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k$ сеть переходит в правильное /нормальное/ состояние $\bar{y} = (y_1, y_2) = (u, 0)$, а в случае наличия ошибки $7(0)$ в результате этой же последовательности получается ошибочное состояние $\bar{y} = (u,1)$. Ошибочное состояние можно истолковать и таким образом, что присутствует виртуальная ошибка $y_2(1)$

Для решения задачи попытаемся распространить воздействие $y_2(1)$ и $7(0)$ к выходу z_1. Это, в свою очередь, достигается таким образом, что на комбинационной части сети выполняется двойное согласование. Входной вектор $\bar{x} = (0, 1, 1)$, полученный в качестве результата согласно рис. 6, соответствует требованиям. Если этот вектор подключается к уже существующей входной последовательности длиной k, тогда получается последовательность тестов длиной $k+1$ для ошибки $7(0)$.

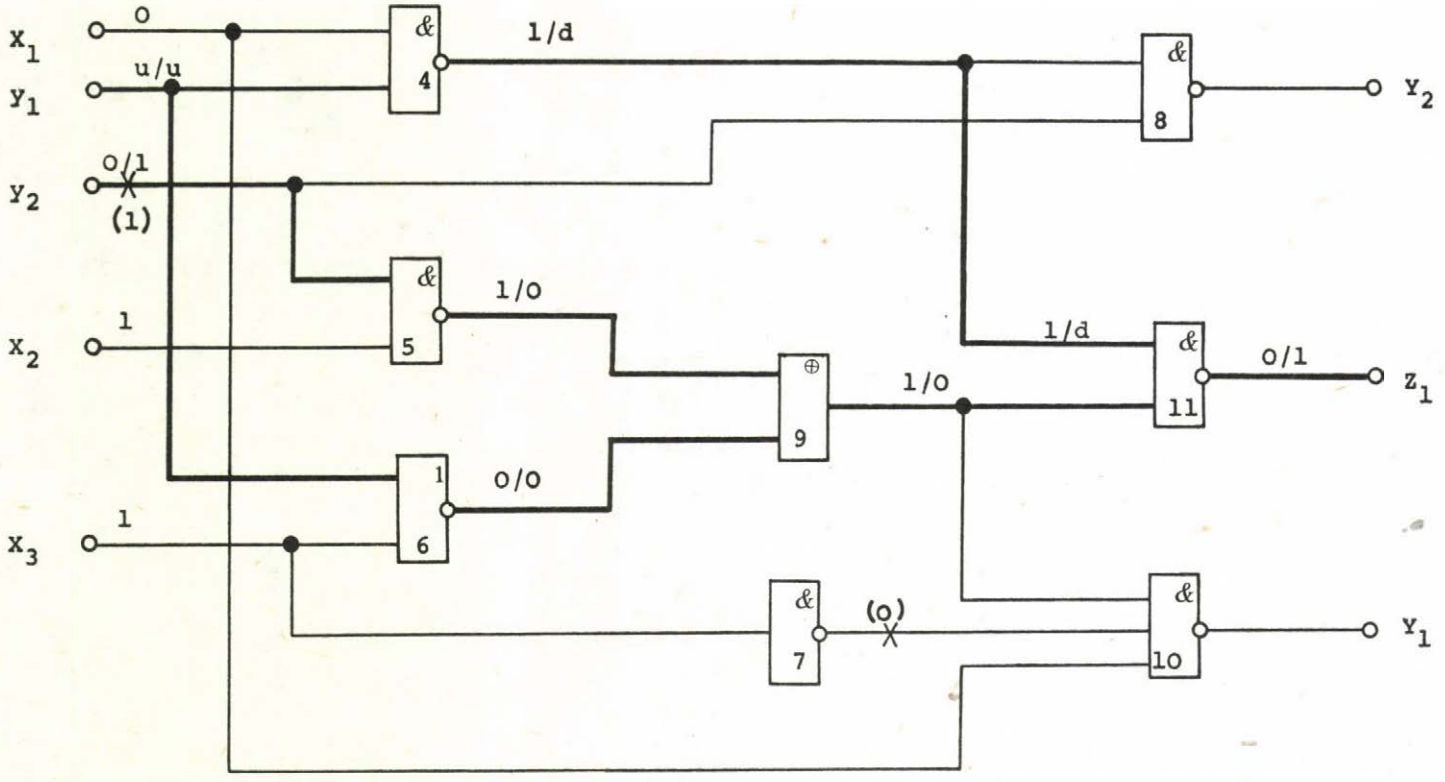


Рис. 6. Вычисление тестов для синхронно-последовательной сети

5. ОЦЕНКА И СРАВНЕНИЕ

В настоящем разделе дается краткое сравнение метода двойного согласования с алгоритмом D и методом булевских разностей. С точки зрения вычислений для этого достаточно учитывать единственный выход обнаружения ошибки, пусть этим выходом будет z_j .

В начале рассмотрим одиночные ошибки. Известно, что алгоритм D состоит из двух фаз вычислений: в первой фазе задаются логические значения для распространения ошибки по какой-то комбинации путей, а во второй фазе он пытается выполнить согласования этих значений. Если последнее не удается, тогда алгоритм находит новую комбинацию путей распространения ошибки и т.д. В самом неблагоприятном случае обрабатываются все возможные комбинации путей распространения, и вслед за ними выполняется согласование. В работе [6] рассмотрен значительно улучшеный вариант алгоритма D, согласно которому вместо обработки всех возможных комбинаций путей требуется всего лишь обработка возможных индивидуальных путей распространения. Однако эта модификация также требует применения фазы распространения ошибки и следующего за ней процесса согласования, причем его необходимо выполнить всегда для всей сети в целом.

По сравнению с этим подходом алгоритм двойного согласования полностью исключает попытки по распространению ошибки и в нем выполняется всего лишь единственный процесс согласования, в результате которого в конце его выполнения автоматически выдается комбинация путей, по которым ошибка распространяется. Поэтому наличие так называемых реконвергентных путей [3] значительно меньше влияет на данное решение, чем хотя бы в случае модифицированного алгоритма D.

Метод булевских разностей или любой его вариант в случае сетей значителных размеров требует формирования и инвертирования огромного количества функций Буля. С другой стороны, для решения различных уравнений алгебры логики требуется выполнить

огромное количество алгебраический операций. Другая трудность заключается в том, что при этом потребность в памяти также может оказаться слишком большой, причем ее довольно трудно оценить заранее. Это, в свою очередь, как раз противоположно методам, основанных на обработке логических значений, для которых потребность в памяти является минимальной. Например, для хранения 16-и логических значений алгоритма двойного согласования достаточно распределить для каждой точки сети по 4-м двоичным разрядам. Просто программируемые и легко алгоритмизируемые вычисления приводят к тому, что последнее решение намного лучше можно приспособить для реализации на ЭВМ.

В случае многократных ошибок преимущества двойного согласования еще лучше выражаются. При этом для $q>1$ ошибок в самом неблагоприятном случае как при алгоритме D, так и методом булевских разностей требуется повторить (2^q-1) -раз, причем с некоторым изменением, те вычисления, которые были необходимы в случае одиночной ошибки [3]. Даже при модифицированном варианте алгоритм D согласно работе [6] может потребоваться повторение вычислений в q раз. По сравнению с этим в алгоритме двойного согласования независимо от количества многократных ошибок всегда требуется один и тот же процесс вычислений, который даже в самом неблагоприятном случае необходимо повторить всего лишь два раза, при перестановке первоначальных значений z_j.

По сравнению с другими методами все это обеспечивает возможность более выгодной реализации алгоритма, обладающей на практике большей живучестью, что особенно в области последовательных сетей имеет большое значение. Это подчеркивается нашими опытами, полученными до сих пор после составления машинной программы данного алгоритма. Программа, составленная на языке Ассемблер, входит в состав системы программ, генерирующих тесты, названной TGP-15 и разработанной на ЭВМ модели 7755 фирмы Сименса Института по координации вычислительной техники /ИКВТ/.

Для последовательных сетей уровня БИС применение алгоритма предполагаемо будет выгодным тогда, если схемы были разработаны по правилам, значительно облегчающим тест-контроль. Так называемая система **Level-Sensitive Scan Design /LSSD/**, внедренная фирмой **IBM**, включает в себя такие, между прочим чрезвычайно положительные правила [7]. Как известно, задача формирования тестов для последовательных схем, разработанных согласно требованиям **LSSD**, упрощается до уровня проектирования тестов для комбинационных сетей. В этом случае целесообразно применять вариант алгоритма двойного согласования для уровня модулей.

СПИСОК ЛИТЕРАТУРЫ

[1] J.P. Roth: Diagnosis of automata failures: A calculus and a method, IBM Journal of Research and Development, Vol. 10. pp. 278-291, July 1966.

[2] F.F. Sellers, M.Y. Hsiao and C.L. Bearnson: Analysing errors with the Boolean difference, IEEE Trans. on Computers, Vol. C-17, pp. 676-683, July 1968.

[3] M.A. Breuer and A.D. Friedman: Diagnosis and Reliable Design of Digital Systems, Computer Science Press, Inc., 1976.

[4] Sziray J.: Logikai hálózatok diagnosztikája, Kandidátusi értekezés, Budapest, 1976.

[5] Sziray J.: A test calculation algorithm for module-level combination networks, Digital Systems Lab, Stanford University, Stanford, California, Technical Note, No. 130, November 1977.

[6] P. Muth: A nine-valued circuit model for test generation, IEEE Trans. on Computers, Vol. C-25, pp. 630-636, June 1976.

[7] E.E. Eichelberger and T.M. Williams: A logic design structure for LSI testability, Proc. 14-th Design Automation Conference, pp. 462-468, New Orleans, June 1977.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

IRREDUNDANT LOGIC CIRCUIT DESIGN BY DECOMPOSITION

Krzysztof Walczak and Krzysztof Sapiecha

Institute of Computer Science
Technical University of Warsaw

General notations

The following notations are used in this paper:

Boolean disjunction : v or $\bigvee_k$

Boolean conjuction : $\cdot$ or $\bigwedge_k$

Boolean negation : $'$

Modulo two sum : $+$

Boolean exponentiation: $x^e = \begin{cases} x & \text{if } e=1 \\ x' & \text{if } e=0 \end{cases}$

Combinational circuit realizing function F : $\hat{F}$

Partial derivative : $\frac{\partial F}{\partial x_i}$

where $\frac{\partial F}{\partial x_i} = F(x_1,\ldots,x_i,\ldots,x_n)+F(x_1,\ldots,x_i',\ldots,x_n)$

Vector of value of variables belonging to a set A which are defined by binary notation of number i: a^i

Single-fault-test set of circuit $\hat{H}$: T_H

1. INTRODUCTION

The well known fact is that detection tests derived by single-fault analysis might not be valid for redundant circuits. To make a test set valid it is necessary to verify that every test remains valid if preceded by any sequence of undetectable faults. Since it requires an extensive analysis to determine second-generation redundancies, this becomes a very difficult problem. The more desirable alternative is to eliminate all redundancy.

This paper presents a method of irredundant logic circuits design by decomposition. The method consists in testing whether a circuit is redundant that is based upon the following definition given by Fridrich and Davis [2].

Definition 1.

A combinational network is irredundant if it is possible to detect all permanent s-a-0 and s-a-1 faults within the network.

2. LINKAGE REDUNDANCY

Let's consider a simple decomposition F(X)=G(H(A,B)B,C). In Figure 1 the circuit realizing this decomposition is shown. Let's assume the circuits $\hat{G}$ and $\hat{H}$ are irredundant.

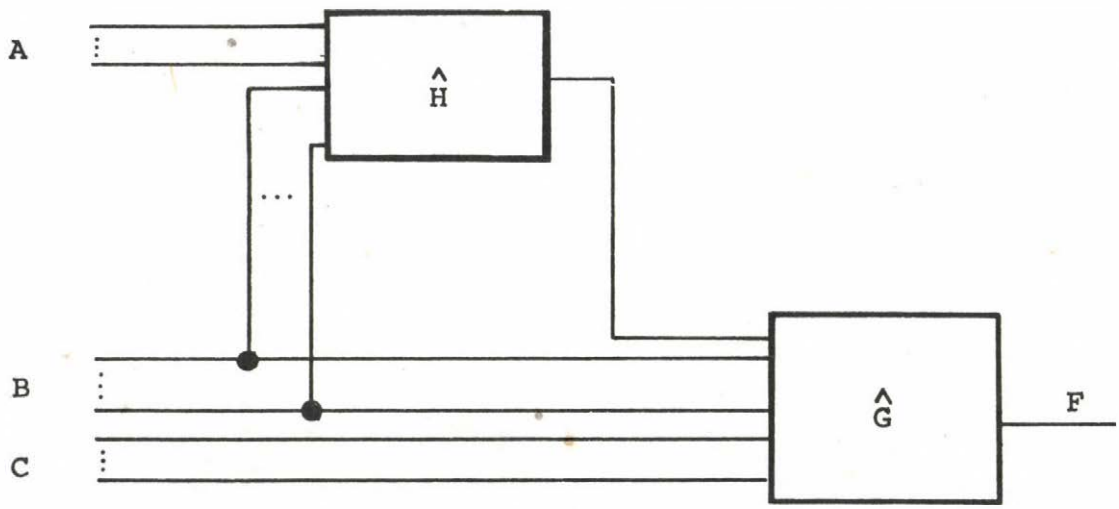


Fig.1.

It means that all the failures within the circuits $\hat{G}$ and $\hat{H}$ are detectable by single-fault-test sets T_G and T_H respectively. The question arises whether this property is hold for composed circuit $\hat{F}$. Generally the answer is not that is illustrated in Figure 2. The a/0 failure is undetectable because of redundancy of the line "a". Redundancy of the above type will be called a linkage redundancy and defined as follows:

Definition 2.

A circuit realizing a simple decomposition $F(X) = G(H(A,B)B,C)$ contains a linkage redundancy if circuit $\hat{F}$ is redundant despite irredundant realization of circuits $\hat{G}$ and $\hat{H}$.

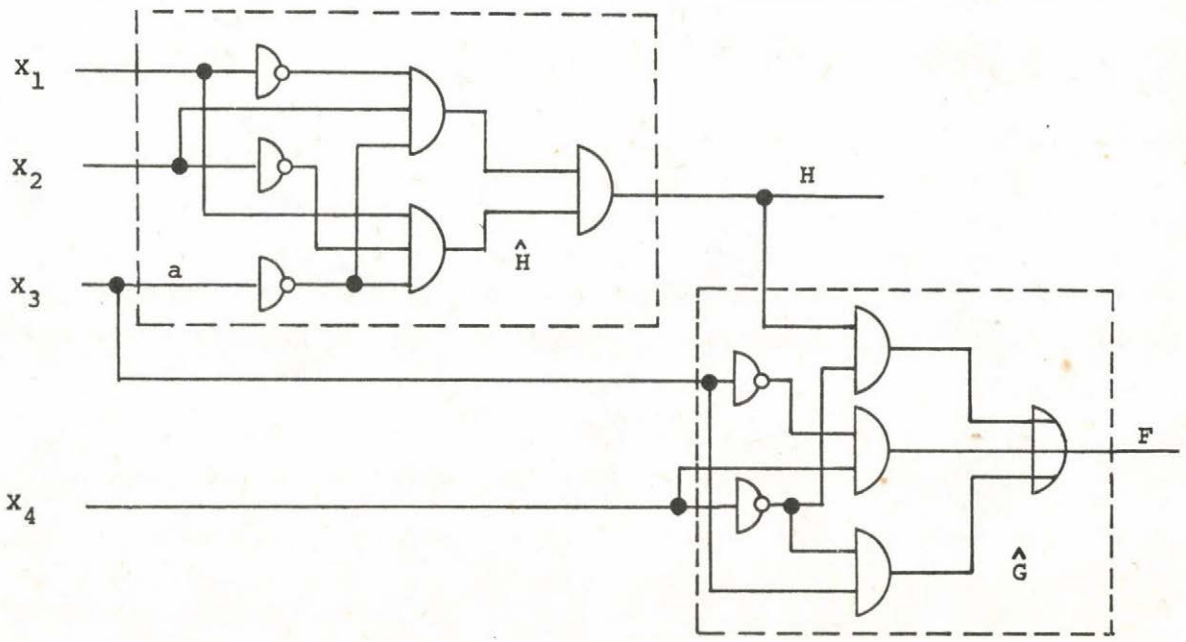


Fig.2.

The above example shows the harmful influence of considered redundancy upon testability of a circuit. In the following we would try to eliminate such redundancy or even better to formulate a method of irredundant realization of decomposed circuit $\hat{F}$.

Realization of the circuit $\hat{F}$ is defined by two functions G and H. A number of realizations of a decomposition F(X)=G(H(A,B)B,C) lays in the range:

$$<2^{2^{card\,B}}, (2^{2^{card\,A}})^{2^{card\,B}}>$$

Definition 3.

Realization of a decomposition contains a linkage redundancy if circuits $\hat{G}$ and $\hat{H}$ do not exist which realize functions G and H such that a circuit $\hat{F}(X)=\hat{G}(\hat{H}(A,B)B,C)$ is free of linkage redundancy.

In the next section the question whether among all realizations of decomposition there is such realization which is free of linkage redundancy is answered.

3. IRREDUNDANT DESIGN METHOD

Our goal is to find out a method of choosing a proper realization that is a linkage redundancy free realization of simple nonsubset* decomposition given. It is possible to derive all realizations of the simple decomposition F(X)=G(H(A,B)B,C) by the array M which is a modified truth table of the function F. The columns of this array are depicted by values of variable from sets B and C and the rows of the array are depticted by values of variables from sets B and C. In the array M the

---

* Simple decomposition is called nonsubset if it is not possible to decrease the set B.

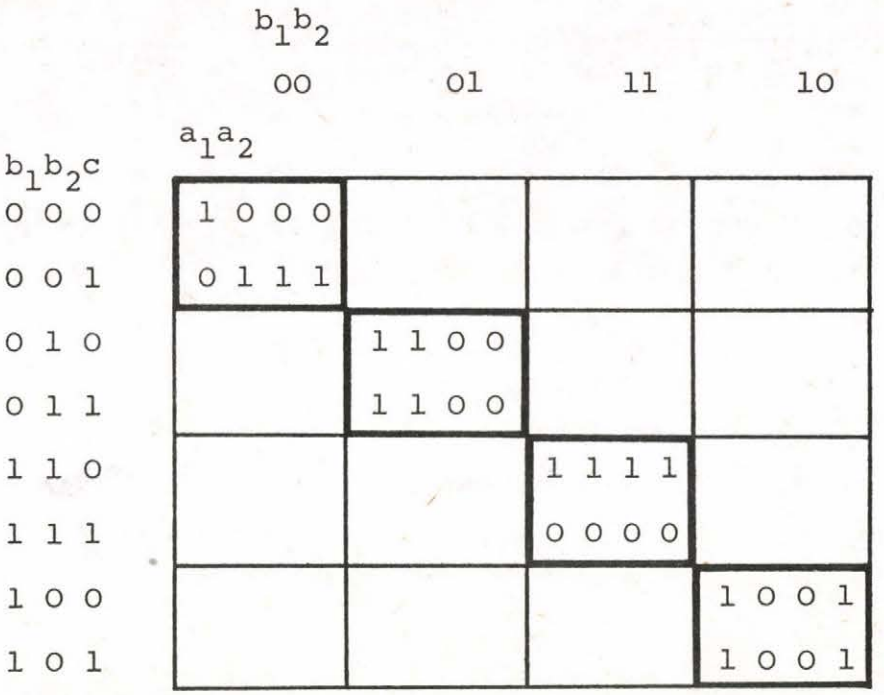


Fig.3.

blocks which are Karnough tables for functions $F(A,b^i,C)=$ $=F^i(A,C)$, $i=0,\ldots,2^{cardB}-1$ are distinguisged. A block will be called a trivial block if it contains only trivial rows. A row is called a trivial row if values contained in the row are "all ones or zeros". For nontrivial block there are two possibilities of how to choice the function H^j (where $H^j(A)=H(A,b^j)$).

It results from the fact that in a block there can exist at most two different nontrivial rows (one is a completion of another). However, in trivial block the function H^j can be any function of cardA variables. It is why the number of all realizations of a given decomposition $F(X)=G(H(A,B)B,C)$ fulfills inequality given in the previous section.

Let us consider the array M which does not contain trivial blocks. In this case all realizations of a given decomposition are linkage-reduncancy free. It results from the fact that all single stuck-at faults in the circuit $\hat{H}$ are detectable, because for all b^i exists c^j such that

$$\left.\frac{\partial G}{\partial H}\right|_{b^i \quad c^j} = 1$$

It means that the output of the subcircuit $\hat{H}$ can always be sensitized to the output of the cirucit $\hat{F}$. Likewise all faults in the circuit $\hat{G}$ are detectable because for all b^i function $H(A,b^i)$ is not equal constant.

If the array M contains trivial blocks then

a/ every test $t \in T_H$ would not contain a value b^j that suited a trivial block (test $t=(a^i b^j)$, where the value b^j suits trivial block will be called forbidden test).

b/ It would be possible to input all tests $t \in T_G$ to the circuit $\hat{G}$.

In summary one can observe that realization of decomposition would meet the two following requirements:

1. Every test $t \in T_H$ would not be a forbidden test.
2. The function H in trivial block would not be a constant function.

It is readily seen that the second requirement can be meet immediately without any troubles. Now we show that the first requirement can always be fulfilled by a sum of product realization of the function H. It is obvious that a forbidden test is not necessary for testing faults on the inputs of the circuits H. It results from the fact that the value b^i suited a nontrivial block such that $\left.\frac{\partial H}{\partial a_j}\right|_{b^i} = 1$ and the value b^j suited a nontrivial block such that $\left.\frac{\partial H_k}{\partial b}\right|_{b^j} = 1$ always exist. However, the conditions which must be satified by the realization of the function H so that all internal faults in circuit H were tested without forbidden tests should be considered.

Let

$$H = x_{ij} p_0 \vee \bigvee_k p_k$$

where p_k are simple implicants of function H and x_{ij} are branches of fanouted line x_i.

All tests of s-a-0 and s-a-1 faults on line x_{ij} can be obtained by the use of the two expressions:

$$t^0_{ij} = p_0 \prod_k p'_k \Big|_{x_i=0} \qquad (1)$$

$$t^1_{ij} = p_0 \prod_k p'_k \Big|_{x_i=1}$$

Then the following conditions must hold:

C1: both the expressions (1) are free from elementary products $a_1^{d_1} \ldots a_{k_n}^{d_{k_n}} b_1^{e_1} \ldots b_m^{e_m}$ in which vector $e_1,\ldots,e_m$ suits a trivial block,

C2: both the expressions (1) are free from elementary products $b_1^{e_1} \ldots b_m^{e_m}$ in which vector $e_1,\ldots,e_m$ suits a nontrivial block adjacent to a trivial block

C3: every product $\prod_k p_k$ can be equal (1) only for the values of variables from the set B which are not suited a trivial block.

Conditions C1 and C2 mean that in Karnough table* there cannot exist any implicant which contains himself only in rows depicted by the values which are suited a trivial blocks or a nontrivial block adjacent to a trivial block. Condition C3 means that for every simple implicant $x_{ij}p_0$ in cutting of Karnough table depicted by $p_0=1$, value zero must exist in a row which is suited a nontrivial block. It fits the situation when the line x_{ij} is stuck-at-1. If the line x_{ij} is stuck-at-0 the implicant x_{ij} has value zero. Then the values "one" of the implicant $x_{ij}p_0$ in the rows suited a nontrivial block cannot be covered by any other implicant of the function H.

It is obvious that conditions C1,C2,C3 always can be fulfilled by suitable choice of the function H in a trivial block.

---

* In considered Karnough table, the rows are depicted by variables from the set B and columns are depicted by variables from the set A.

Conclusion. Nonredundant realization of nonsubset simple decomposition $F(X)=G(H(A,B)B,C)$ can always be obtained by the fulfilment of the conditions C1, C2, C3 and by the realization of the function H as a sum of products.

Example. Let us consider nonsubset simple decomposition $F(X)=G(H(a_1,a_2,b_1,b_2),b_1,b_2)$, for which the array M is presented in Figure 3. Karnough table of the function H_1 which fulfills conditions C1, C2, C3 is presented in Figure 4. The circuit realizing the function H is shown in Figure 5. The forbidden tests are not necessary to testing of the circuit $\hat{H}_1$. For example single-fault-test set T_H is a follows:

$$T_H=\{0000,\ 0100,\ 1000,\ 0101,\ 1101,\ 1010,\ 1110\}$$

b_1b_2 \ a_1a_2	00	01	11	10	
00	1	0	0	0	
01	1	1	0	0	
11	1	1	0	1	← the row which is suited a trivial block
10	1	0	0	1	

$$H = a_1'a_2' \vee a_1'b_2 \vee a_2'b_1$$

Fig. 4.

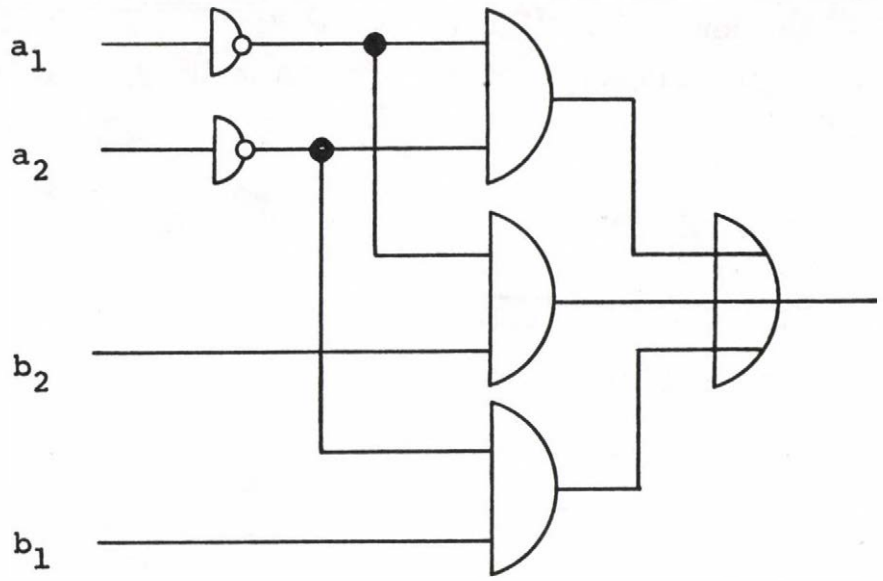
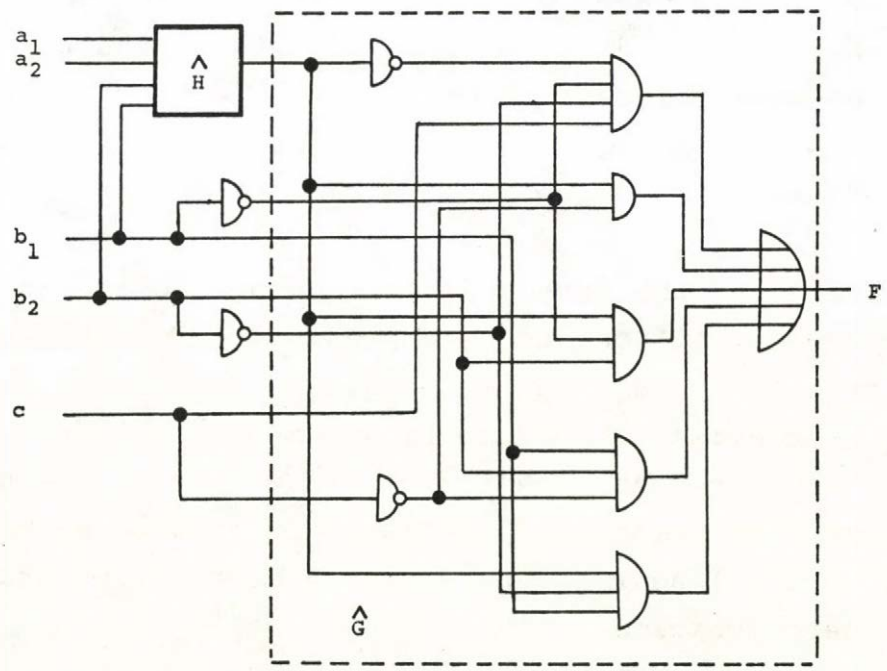


Fig.5.

This set does not contain the forbidden tests and therefore the circuit $\hat{F}$ is linkage redundancy free (Figure 6).



$G = H'cb_1'b_2' \vee Hc' \vee Hb_1'b_2 \vee b_1b_2c' \vee Hb_1b_2'$

Fig.6.

However, if the function H is chosen in another way, for example as presented in Figure 7, the circuit $\hat{F}$ would contain redundancy by realization of the function H as a sum of products.

b_1b_2 \ a_1a_2	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	0	0	1	0
10	1	0	0	1

$$H_2 = a_1a_2b_1b_2 \vee a_1'a_2'b_2' \vee a_1'a_2'b_1' \vee a_1'b_1'b_2 \vee a_2'b_1b_2'$$

Fig.7.

However, this paper does not decide, if another realization of the function H exists such that the circuit $\hat{F}$ would be linkage redundancy free.

4. CONCLUSIONS

In the paper the method of irredundant logic circuit design by decomposition has been presented. For this purpose linkage redundancy has been defined. The sufficient conditions so that the realization of simple decomposition would be linkage redundancy free have been given. However, in the paper (8) it is shown that the method presented in this paper can be extended to the linear and nonlinear decomposition and to multiple decomposition.

REFERENCES

[1] Curtis H.A.: A new approach to the design of switching circuits. Van Nostrand, Princeton 1962.

[2] Fridrich M., Davis W.A.: Minimal fault tests for combinational networks. IEEE Trans.Comp ., August 1974.

[3] Friedman A.D.: Fault detection in redundant circuits. IEEE Trans. on Electronic Computers, 1967, EC-16.

[4] Lee H.S., Davidson E.S.: Redundancy testing in combinational networks. IEEE Trans. Comp., October 1974.

[5] Shen V.Y.: A fast algorithm for the disjunctive decomposition of switching functions. IEEE Trans. Comp., vol.C-20, March 1971.

[6] Reddy S.M.: Easily testable realizations for logic functions. IEEE Trans. Comp., November 1972.

[7] Thayse A.: A fast algorithm for the proper decomposition of boolean functions. Philips Res.Rep., vol.27. 1972.

[8] Walczak K.: Testing of decompositional circuits. Archiwum Automatyki i Telemechaniki, No.2, 1980.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

Terplán, Sándor

A DESIGN METHOD OF ASYNCHRONOUS SEQUENTIAL CIRCUITS BASED ON FLOW DIAGRAM

Technical University of Budapest
Department of Process Control

1. Introduction

In the solution of practical problems asynchronous sequential networks must be designed in cases when the operational speed expected from the network cannot be achieved otherwise. In usual cases the microprogrammed networks of synchronous operation are the main tools of solving sequential control problems. However, with given IC technology - e.g. bipolar TTL - wired asynchronous logics always permit a faster operation than the microprogrammed technologies. With these latter the tasks to be performed simultaneously are divided into elementary steps, and the required operation is accomplished by successive steps. This fact justifies even today the application of asynchronous control units in fast working equipment. Such are, e.g. the high-speed mass stores of computers, as well as their interfaces, some real-time acquisition devices, telecommunication and video systems, some real-time physical simulation system, etc.

The classical asynchronous designing procedures starting from a flow table are less applicable to the design of practical control units, since the flow table becomes puzzling by its vast dimensions. At the same time, the necessity of testing for essential hazards and critical races makes the designing work complicated.

In recent years new procedures have been reported in literature,

which try to solve the above mentioned problems. A group of the procedures is based on the application of elementary asynchronous edge-sensitive sequential networks. Such edge-sensitive elements are, e.g., the M and G elements [7], the I-T flip-flop and the D-TSFF [9].

To describe the operation, various "edge-sensitive" flow tables are used which can be regarded as developed forms of the classical flow table. The edge-sensitive elements exclude the possibility of essential hazards and critical races, therefore they employ the coding procedures applicable with synchronous networks. Howevers, as has been shown by E. SCHMITT and S.WENDT [12], a proper operation is not guaranteed even with synchronous sequential networks, if arbitrary coding and, e.g., edge-sensitive D or J-K flip-flops are used. Therefore, handling the hazard phenomena in this manner does not yield the final solution. As has been pointed out by S. BEISTER [13], guaranteed proper operation can be achived, if the paths of signal transmission within the logical network are taken into account separately in the design work even in the case when a path consists of wires only. A detailed analysis and development of the "edge-sensitive design methods" is given in the work of R. KIRCHNER [10]. Another way is the procedure recommended by R. DAVID [11], who uses a state graph for the description of the task. An elementary asynchronous sequential network, the so-called CUSA /Cellule Universelle pour Séquences Asynchrones/ is assigned to each state. Choosing "1 out of N" to code the state, the CUSA realizes the product of the disjunction of the secondary variables and of the input combination. The author requires certain formal properties from the state graph assumed by the designer, and has elaborated a systematic verification procedure to check their presence. The essential hazard and the critical race condition are eliminated by the delaying conditions of the CUSA built in in advance.

The two basic ideas that can be derived from the above descriptions and can be used also in the present work are as followes:

- Application of flip-flops with edge-sensitive inputs as universal building blocks to the design of asynchronous sequential networks
- Use of a graph insted of the flow table.

At the Department of Process Control at the Technical University of Budapest succesful investigations have been carried on in the field of systematically designing control units of synchronous nature with the use of flow diagrams [1 , 2 , 3] . The systematic procedure suitable for designing asynchronous logical networks to be presented in the paper is linked with these investigations.

2. Symbols of the asynchronous flow diagram for the case of changes in adjacent input combinations

The flow chart is assumed by the designer intuitively on the basis of worded or other conditions. The operational sequence is separated into so-called phases. The phase is a concept similar to state. The phase transitions are triggered by the changes of the input combinations important from the view-point of the output sequence. These changes are stated by the designer in the so-called triggering flow diagram instructions in the form of functions F interpreted on the input signals. The transition 0 - 1 taking place in the value of function F indicates the phase transition. The developing new phase is determined by the "yes" branch of the triggering instructions, while the existing phase is generated by the triggering instructions connected with each other through the "no" branches /the sequence is irrelevant/. The flow chart is normal with respect to the phases, i.e., a given change of the input combination can cause only one direct phase transition. The functions F taking part in bounding the developing new phase may have arbitrary value, and even a transition 1 - 0 may take place in the value of these functions F at the time of the development of the new phase, since the 1 - 0 transition never generate a phase

transition.

If the designer finds such a condition formed from the input signals that can exclusively enable the respective phase transition, then he may state this in so-called branching instructions in the form of function G interpreted on the input signals. Thus a condition or enabling function G can be assigned to each of functions F. The 0 - 1 transition taking place at the output of function F means a phase transition in the case only if, during this change of the input combination, function G has a logical value "1". If the 0 - 1 transition occurring at the output of function F does not mean a phase transition, then the logical value of function G is "0" during this change of input combination.

Assignment of a function F and a function G to each other takes place in the way that the YES branch of the triggering instruction corresponding to function F leads to a so-called branching instruction in which function G is defined in the form $G = f(x) = 1$. The YES branch of the branching instruction, which means the validity of $G = 1$, leads to the next phase. The "NOT" branch of the branching instruction leads to the same place as the NOT branch of the triggering instruction, i.e. to some other instruction pair taking part in the determination of the existing phase. The two NOT branches together, quasi "connected parallel" take part in the determination of the just existing phase. If two functions F leading to different next phases agree with each other and their functions G are the opposite of each other, then a simpler notation can be used in the flow diagram. The two functionpairs F and G can be replaced by one. In such a case, also the NOT branch of the branching instruction leads to a next phase.

If the Moore model is applied, the output combinations can be produced with the aid of the functions formed from the out-put of the phases, i.e., the value of the output signals is assigned to the phases. This is simply done if the phases have

codes "1 out of n". The Moore model network structure corresponding to the flow diagram discussed so far is shown in Fig. 1.

In fact, the phases mean such a partition of the states of a logical problem given by a normal, asynchronous primitive flow table according to the Moore model, where each block corresponds to a phase and they have the following properties:

a./ No contradictory output combinations belong to the states being in the same phase.

b./ In each column of a flow table portion corresponding to the individual phases there cannot be more than one stable state entry.

c./ The columns of each adjacent input combination pair x^k, x^l, where the change of the input combination $x^k \rightarrow x^l$, produces during the phase transition a stable state belonging into the phase, satisfy the following condition: There is no row in which there would be a stable state entry under x^k during the time when under x^l there is an entry meaning a state belonging to some other phase.

With symbols:

If $f_q(x^k, y_i) = y_i$

$f_q(x^l, y_i) = y_j$

$f_q(x^l, y_j) = y_j$

where $y_i \in Q_q, y_j \in Q_q$, then there is no such a row in phase Q_q in which

$f_q(x^k, y_r) = y_r$

$f_q(x^l, y_r) = y_p$

$f_q(x^l, y_p) = y_p$ where $y_r \in Q_q$, for all indices r coming into question, and $y_p \in Q_p, Q_q \neq Q_p$

The partition defined in this way constitutes the basis to the procedures of conversion of flow table into flow diagram and of flow diagram into flow table, as detailed in paper [4]. With this, it can be demonstrated that the flow diagram describes the operation uniquely in the case only when the base states of the network are known. Therefore they must be stated separately.

Following from the above partition, the statement of the base state means the statement of the base phase and of the input combination.

As the code "1 out of n" has been chosen as code of the phases, then the inpermissible danger arises that the output signals having identical logical value for several phases change in the transient at phase transition. At the same time it is advisable to transform the structure according to the Moore model used so far into a Mealy model so as to decrease the number of the states.

For the two above reasons, the network part producing the output signals must be omitted from the block diagram shown in Fig. 1, and the network part shown in Fig. 2 has to be used instead.

In this way the following instructions setting the output signals have been introduced:

a./ SET instruction

The output signal to which it refers will have a logical value "1" if function K_s exists or arises in the given phase of the network. After this, the value of the output signal will remain unchanged until the next RESET instruction pertaining to the output signal.

b./ RESET instruction

The output signal to which it refers will have a logical value "0" if function K_R exists or arises in the given

phase of the network. After this the value of the output signal will not change until the next SET instruction pertaining to the output signal.

c./ D instruction

The output signal to which it refers follows the variation of function K_D in the given phase of the network, then holds the value existing at the moment of the emergence of the next phase until the same instruction comes on.

d./ Function instruction /K_1 type, maxterm form/

The output signal to which it refers, follows the changes of function K_1 in the given phase of the network, then, after cessation of the phase, will have the logical value "1" until the function instruction relative to the output signal /until type K_1/

e./ Function instruction /K_0 type, minterm form/

The output signal to which it refers, follows the variation of function K_0 in the given phase of the network, then, after cessation of the phase, will have the logical value "0" until the next function instruction /of type K/ relative to the output.

The instructions setting the output signals have to be placed in the phases, in the part bounded by the trigger instructions. In this way they will be assigned to the phases.

3. Properties of the functions included in the flow diagram

Functions F and G have the properties listed below:

a./ The output of any function G remains unchanged in the just existing phase, if a transition 0 - 1 takes place on the output of function F belonging to it.

b./ Of any just existing phase it is true that out of those functions F taking part in determination of the phase whose pair functions G are just enabling some phase transi-

tion always only on the output of one single function can a transition 0 - 1 take place upon the effect of the given change of input combination. To functions F of the trigger instructions without branching instruction belongs a function G with value G = 1.

c,/ If in any phase just existing, a transition 0 - 1 indicating a phase transition arises on the output of any of the functions F determining the phase, then no transition 0 - 1 may arise on the output of any of those functions F taking part in the determination of the thus prescribed next phase whose pair functions G are just enabling some phase transition. To the functions F of the trigger instructions without branching instruction belongs a function G with value G = 1.

Provided that functions F and G of the flow diagram have the above listed properties, functions K in the instructions setting the output signals will have the following properties:

a./ The output of the respective function K_D does not change in the case of any instruction D of the flow diagram, when the phase in which it is present is just arising or ending.

b./ No transition 1 - 0 takes place on the output of the respective function K in the case of any function of type Set, Reset and K_0 of the flow diagram, when the phase in which it exists is just arising, and no 0 - 1 transition may take when the phase ends.

c./ No 0 - 1 transition takes place on the output of function K_1 in the case of any K_1-type instruction of the flow diagram, when the phase in which it exist is just arising, and no transition 1 - 0 may take place when the phase ends. The above properties are in fact simple formal requirements against the flow diagram assumed on the basis of worded conditions.

Drawing a parallel with the flow table, properties a. and b.

relative to functions F and G mean that a given flow table cell cannot contain an entry indicating two next states. Property c. means that the flow diagram is normal. The properties relative to functions K exclude the possibility of functional hazards in these functions. Reference [4] states conditions to the check-up of the properties of functions F, G and K of the flow diagram. A test procedure is elaborated in detail /whose algorithm for a computer can be made as well/ by which the tests can be performed systematically.

4. Realization of an asynchronous phase register network for the case of changes in adjacent input combinations

4.1 Realization of functions F and G

The functions are given explicitly in the flow diagram. In their realization the only requirement is the freedom from statical hazards. Then, assuming a two-level realization, no peak may arise on the output of the functions, which would lead to malfunctioning. As the changes of the input combination can only be adjacent, there is no possibility of functional hazards.

4.2. Realization of the output signals

With the assumption that the phases are coded by "1 out of n", the mappings indicated in Fig. 2 can directly be accomplished by two-level logical functions realized free from static hazards.

4.3. Introduction of the phase register structure

If the phases are coded in "1 out of n", then the network structure part characterized by the mapping $f_Q(F,G,Q) \Rightarrow Q$ leads to the network shown in Fig. 3. The control Q_{iv} of the phase register cell is a set of control signals. The phase register cell can be divided to further network parts by a suitable grouping of the control signals present in the set

Q_{iv}. Such a partition can be seen in Fig. 4. To each function F belongs a so-called phase register element, which by itself is an asynchronous sequential circuit with an operation similar to that of the flip-flops. The disjunction of the outputs of these phase register elements is the phase itself. The transition "0-1" taking place at the output of function F sets the phase register element assigned to the function, if the phase, in the formation of which function F takes part, exists and function G belonging to function F enables the phase transition. The occurrence of any phase following the previous phase - i.e. the transition "0-1" taking place in the value of signal Q meaning the phase - erases the phase register element. From the properties of the defined function pairs and from the code "1 out of n" of the phases follows that during the whole operation only one single phase register element is set in the network.

4.3.1. Realization of the phase register element

Based on the phase register resolution according to Fig. 4 and on the flow diagram operation stated in the foregoing, the operation of the phase register as asynchronous sequential network can be described as follows:

Worded condition; Inputs: CKJ, CKK, J
Outputs: q

1./ Output q of the network changes from 0 to 1 if and only if input CKJ changes from 0 to 1 and during this time there is "1" on input J, the value of input CKK is arbitrary and even the transition 1-0 is permitted.

2./ Output q of the network changes from 1 to 0 if and only if input CKK changes from 0 to 1, and during this time input J may change arbitrarily, the value of input CKJ is arbitrary, and even the transition 1-0 is permitted.

A possible realization of the network is given in Fig. 5. This

choice is justified by the fact that a part of the network agrees with the IC D flip-flop /SN 7474/. Instead of D flip-flops also IC edge-sensitive J-K flip-flops, e.g., an SN 7409 J-$\bar{K}$ flip-flop, can be used. In such a case the phase register element is simpler, as the J input is given in advance. The K input has to be connected to logical "1".

With phases coded "1 out of n", the network shown in Fig. 4 is guaranteed to be free from critical races and essential hazards. This is due to the fact that the phase register element as an independent sequential network is constructed free from hazards and critical races, and within the asynchronous phase register obtained by their coupling the individual elements may have a control which is permitted with the proper operation to each element. In some more detail this means that, apart of the changes of adjacent input combinations, only such multiple changes of input combinations may arise at the inputs of the phase register element which do not affect the operation of-the individual phase register elements. If integrated circuits, such as D flip-flops are used as phase register elements, then the asynchronous phase register network giving the solution of the logical problem can be considered to have an operation independent of the speed. In other words: the delay of the individual building elements /gates, flip-flops/ compared with each other does not affect the proper operation to the whole network, since the delay conditions are built into the integrated circuit flip-flops by the factory. After this, the question may arise what happens if the logical problem to be realized does not contain any essential hazard, but at the same time the sequential networks /phase register elements/ as building blocks, obtained during decomposition, always contain it. In such a case the ability of the phase register "to free from hazards" is not utilized. The check-up for essential hazards becomes superfluous just by the fact that - with the phase register structure applied - starting from any state of the network, essential hazards may be encountered, which, however, are always eliminated in the same way.

4.4. An example

The example contains the data transfer block of the control unit connecting a high-speed peripheral having two control signals and a channel IBM 360 working in the selector mode. The data transfer block is part of the channel-side control unit and is subordinated to it.

4.4.1. Operation

Input signals:

ZER:	General signal setting the control unit into basic position
START:	It is displayed by the channel-side control unit when data transfer starts
RDY:	Displayed by the peripheral when it is ready to a new data transfer. An impulse with value "1"
INFRDY:	The peripheral indicates by it the end of the byte transfer. An impulse with value "1"
WR:	Write when its value is "1"; read when its value is "0"
PARF:	When it has a value "1", the byte arrived from the channel and to be written has parity error
UC:	When its value is "1", an error has arisen in the data transfer
SERO:	Service Out: a standard interface signal of IBM 360
CMDO:	Command Out: a standard interface signal of IBM 360

Output signals:

SERI:	Service In: a standard interface signal of IBM 360
INFST:	Its value "1" indicates to the peripheral that

the data transfer block is ready to transfer the next byte

PH1: With its value "1" the data transfer block indicates to the channel-side control unit that it is ready to perform data transfer

The output signals listed below are indications to the channel-side control unit. If any of them has the value "1", it takes the control back from the data transfer block.

PH5: The channel has found a parity error in the reading and therefore has stopped the data transfer by interface disconnect

PH7: DATA transfer stopped /byte counter full/. The channel has responded with Command Out to Service In.

PHS: The control unit has received data with parity error from the channel and therefore the data transfer block stops transmitting data

PH11: The control unit has found an error in data transfer and therefore stops it.

Other conditions:

- The changes of input combinations are always adjacent
- The channel-side control unit raises the signal START in the case only, if PHI has the value "1"
- The data transfer speed of the channel is higher then that of the peripherals, i.e. the signal exchange SERI-SERO is sure to take place between two INFRDY impulses
- In basic position of the data transfer block, the output signal PH1 has the value "1", the others have "0". Then all input signals have the value "0", except for the ZER signal, which has the

value "1", and for the WR signal, which may have an arbitrary value.

4.4.2. Assumption of the flow diagram

The flow diagram described the operation of the network to be designed is shown in Fig. 6. The instructions are identified by their graphic symbols according to their types. The output signals beginning with PH are not given separately in the diagram, they agree with the corresponding phase output.

4.4.3. Formal check-up of the flow diagram

The properties mentioned of the flow diagram functions are sure to be valid in the case of this simple example, for the functions to be examined by pairs are either not interpreted on identical input signals or they are the opposites of each other.

4.4.4. Network realization

The control functions of the phase register of the designed network are shown in Fig. 7.a. The basic states of the network are set by the Preset and Clear inputs of the IC flip-flops of the phase register elements.

The possibility of simplification not seen directly from the structure of the network shown in Fig. 4 is as follows:

If the same function F triggers the network from given phases into the same next phase under the same conditions, then the next phase can be produced by a single phase register element to the J input of which the disjunction of the given phases is connected. This fact could be utilized with phases PH1, PH7 and PH11. With this latter only because also the enabling functions /UC/ of the triggering functions are identical.

The control functions of the output signals are shown in

Fig. 7.b, and the scheme of the network in Fig. 8. For simplicity, the Preset and Clear inputs are not indicated in the Figure.

5. Consideration of multiple input changes

Also the inputs that able to change simultaneously can be taken into account, and, with certain restrictions /e.g. the use of running clock signals, application of auxiliary networks/, there are realization possibilities also for such cases. This is possible by the fact that the J input of the phase register element agrees with the D input of the D flip-flop and its CKJ input with the input of the flip-flop clock signal, and thus the synchronizing property of the D flip-flop can be utilized.

The dissertation [4] discusses also the co-operation of asynchronous phase register networks. To co-operate with equipment built on differing principles, clock signal /synchronous/ flip-flops and monostables, too, can be used in the network part producing the output signals.

6. Conclusions

The steps of the design work can be summarized as follows:

- Assumption of the flow diagram on the basis of worded or other conditions.
- Checking the properties of functions F and G of the flow diagram. If any of them is not present, modification of the flow diagram must be performed based on a new interpretation of the worded conditions.
- Checking the properties of functions K of the flow diagram.

In case of any error, modification of the flow diagram and repeated execution of the previous step have to be performed.

- Based on the flow diagram, according to the network structure shown in Fig. 4, determination of the control functions of the phase register.
- Determination of the control functions of the output part.

The design method discussed has the following features:

- The operation is described by means of a flow diagram, which is the basis of the design.
- The network structure is built on a phase register.
- It is possible to take multiple input changes into account.
- After assumption of the flow diagram, the design work is completely systematic.
- The network is free from essential hazards and critical race conditions, which does not need any special examinations since it is ensured by the Ic-s present in the phase register elements.
- The network realized can be considered independent of speed. Here, the independence of speed means that the signal-delaying effect of the circuits of the network with respect to each other does not affect the proper operation. Of course, the speed of the input signal sequence must not exceed the maximum operational speed determined by the longest path of signal transmission.

REFERENCES

1. Z. László, P. Arató: Microprogrammed Logic Network Design. Mikroprocessors, vol. 2, Bo. 2, April. 1978. p. 73.

2. P. Arató, J. Grantner, P. Kalmár, S. Terplán: Methoden des Logischen Entwurfs auf der Grundlage von Ablaufplänen Z. elektr, Inform.c. Energietechnik, Leipzig 7 /1977/ 5, S. 426

3. Kalmár, P.: A phase- state reduction and assignment method based on the flow chart for the logical design of control units.

4. Terplán, S.: A Method of Designing Asynchronous Logic Networks, Based on Flow Diagram /in Hungarian/. Doctoral dissertation, 1980.

5. S.H. Unger: Asynchronous Sequential Switching Circuits Wiley-Interscience, New York, 1969.

6. E.J. McCluskey: Introduction to the Theory of Switching Circuits
New York: McGraw-Hill 1965.

7. Y.H. Chuang: Transition Logic Circuits and a Synthesis Method
IEEE Transactions on Computers, vol. C-18, No. 2, February 1969. p. 154

8. J.R. Smith, jr., Ch.H. Roth, jr.: Analysis and Synthesis of Asynchronous Sequential Networks Using Edge-Sensitive Flip-Flops

   IEEE Transactions on Computers, vol C-20, No. 8, August. 1971. p. 847.

9. J.K. Bredeson, P.T. Hulina: Synthesis of Multiple - Input Change Asynchronous Circuits Using Transition - Sensitive Flip-Flops

IEEE Transactions on Computers, vol C-22, No. 5. May 1973. p. 524

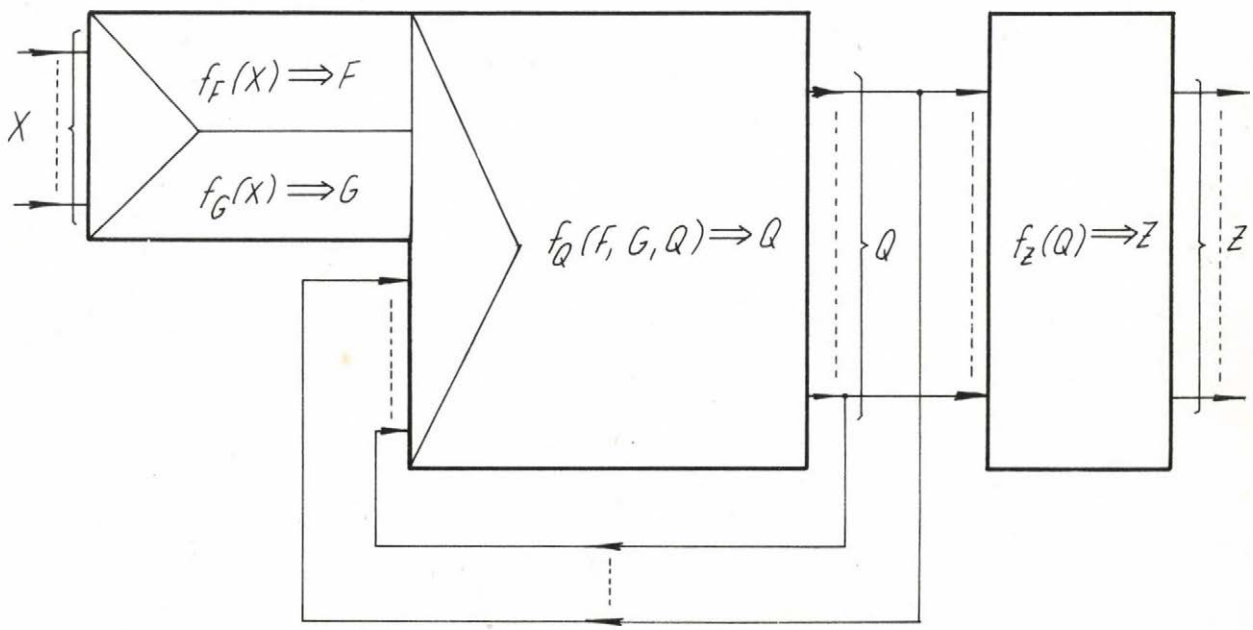
10. R. Kirchner: Asynchrone Schaltwerke mit Flankensteuerung Dissertation zur Erlangung des akademischen Grades eines Doktor- Ingenieurs 1978.

11. R. David: Modular Designing of Asynchronous Circuits Defined by Graphs
IEEE Transactions on Computers, vol. C-26, No. 8, August 1977. p. 727

12. E. Schmitt, S. Wendt: Critical Feedback Analysis of Clocked Digital Systems.
NTZ, Heft 4. 1972. S. 196.

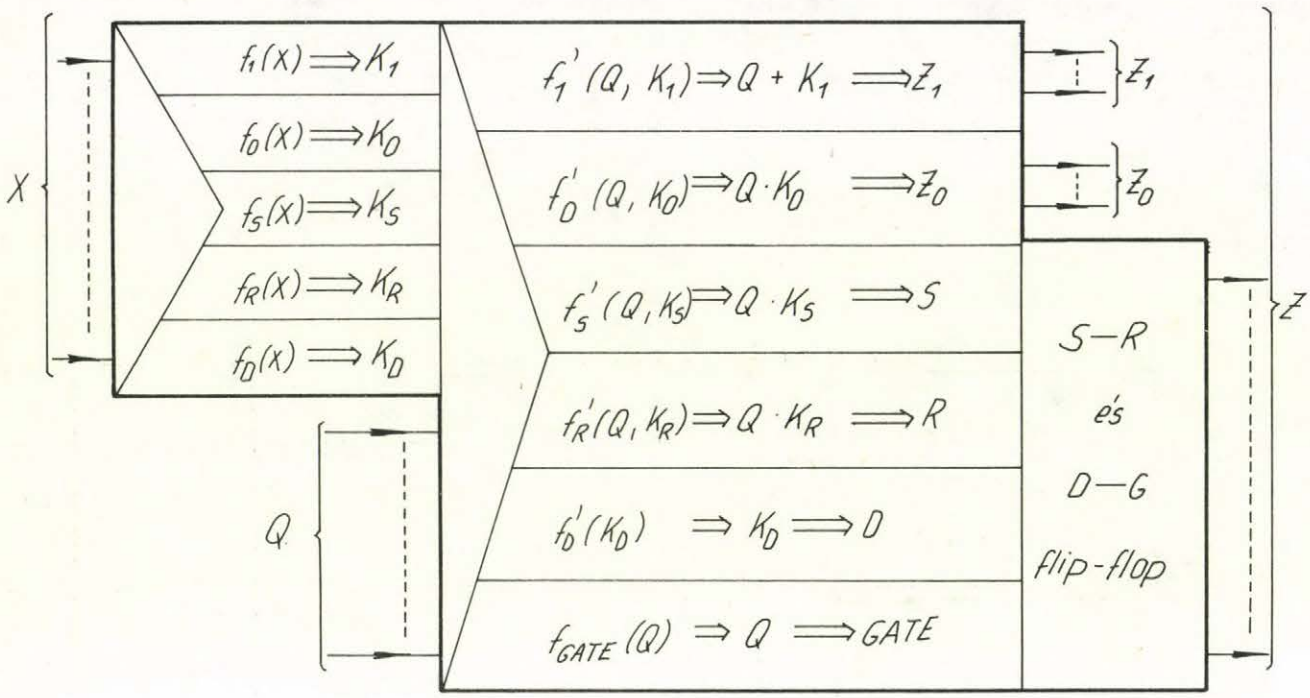
13. J. Beister: A Unified Approach to Combinational Hazards. IEEE Transactions on Computers, vol. C-23, No. 6, June 1974. p. 565

14. J. Beister: Ein Totzeitmodell für asynchrone Schaltwerke NTZ 28, Heft. 1, 1975. S. 13.



X: Set of Input Variables
F: Set of Functions of Trigger Instructions
G: Set of Functions of Branch Instructions
f_F: F Function Mapping
f_G: G Function Mapping
Q: Set of Phases (coded 1 out of n)
f_Q: Next Phase Mapping (in Form of flow diagram)
Z: Set of Output Variables
f_Z: Output Mapping

Fig. 1.



K_1: Set of Functions Type K_1

K_0: Set of Functions Type K_0

K_S: Set of Functions Type „Set"

K_R: Set of Functions Type „Reset"

K_D: Set of Functions Type D

f_1: K_1 Function Mapping

f_0: K_0 Function Mapping

f_S: „Set" Function Mapping

f_R: „Reset" Function Mapping

f_D: D Function Mapping

Z_1: Output Mapping

Z_0: Output Mapping

S: Set of Inputs „Set" of S-R flip-flops

R: Set of Inputs „Reset" of S-R flip-flops

D: Set of Inputs D of D-G flip-flops

GATE: Set of Inputs G of D-G flip-flops

Fig. 2.

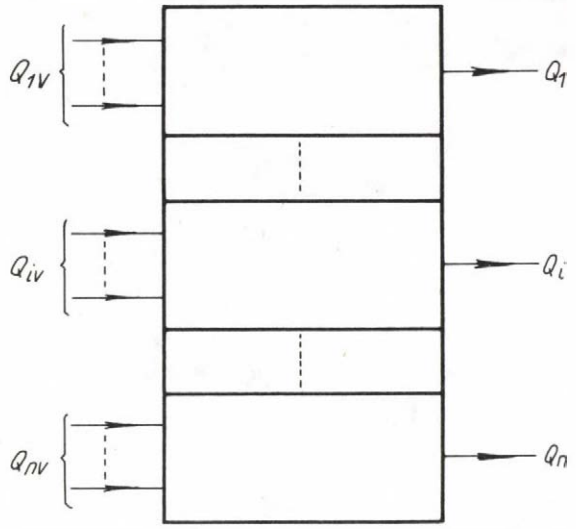

Q_{1V}
Q_1
Q_{iv}
Q_i
Q_{nv}
Q_n

Fig. 3.

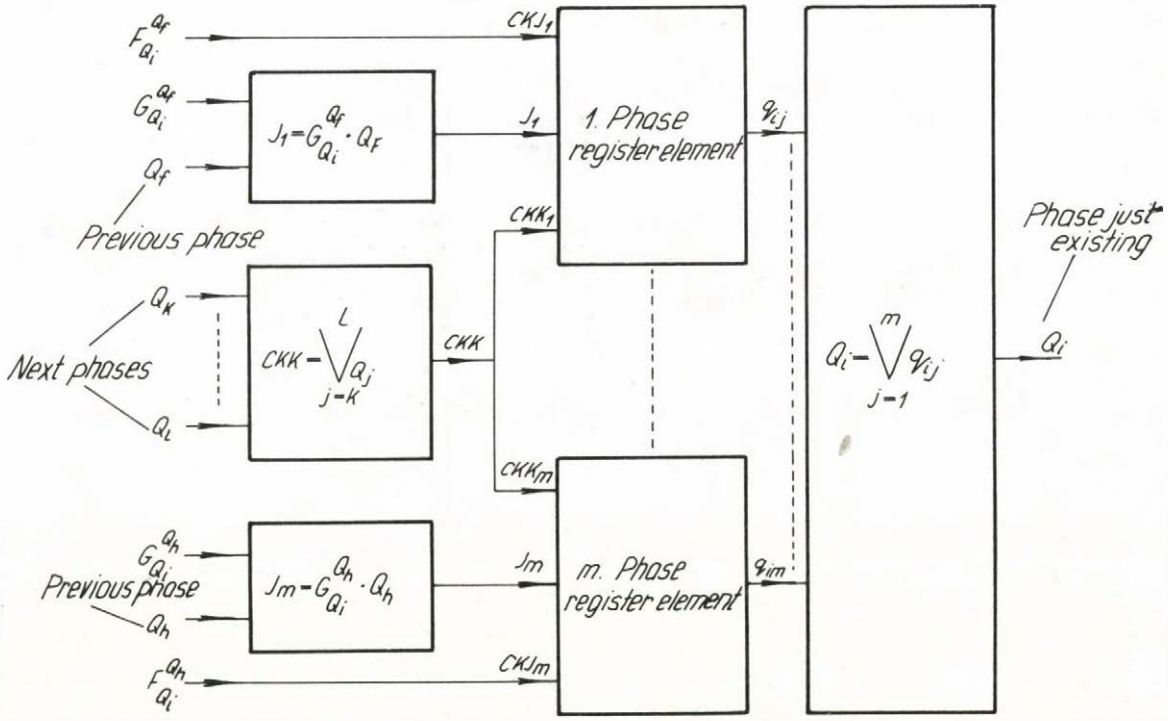

$F_{Q_i}^{Q_f}$
CKJ_1
$G_{Q_i}^{Q_f}$
$J_1 = G_{Q_i}^{Q_f} \cdot Q_f$
J_1
1. Phase register element
q_{ij}
Q_f
Previous phase
CKK_1
Phase just existing
Q_K
Next phases
$CKK = \bigvee_{j=K}^{L} Q_j$
CKK
$Q_i = \bigvee_{j=1}^{m} q_{ij}$
Q_i
Q_L
CKK_m
$G_{Q_i}^{Q_h}$
Previous phase
$J_m = G_{Q_i}^{Q_h} \cdot Q_h$
J_m
m. Phase register element
q_{im}
Q_h
$F_{Q_i}^{Q_h}$
CKJ_m

Fig. 4.

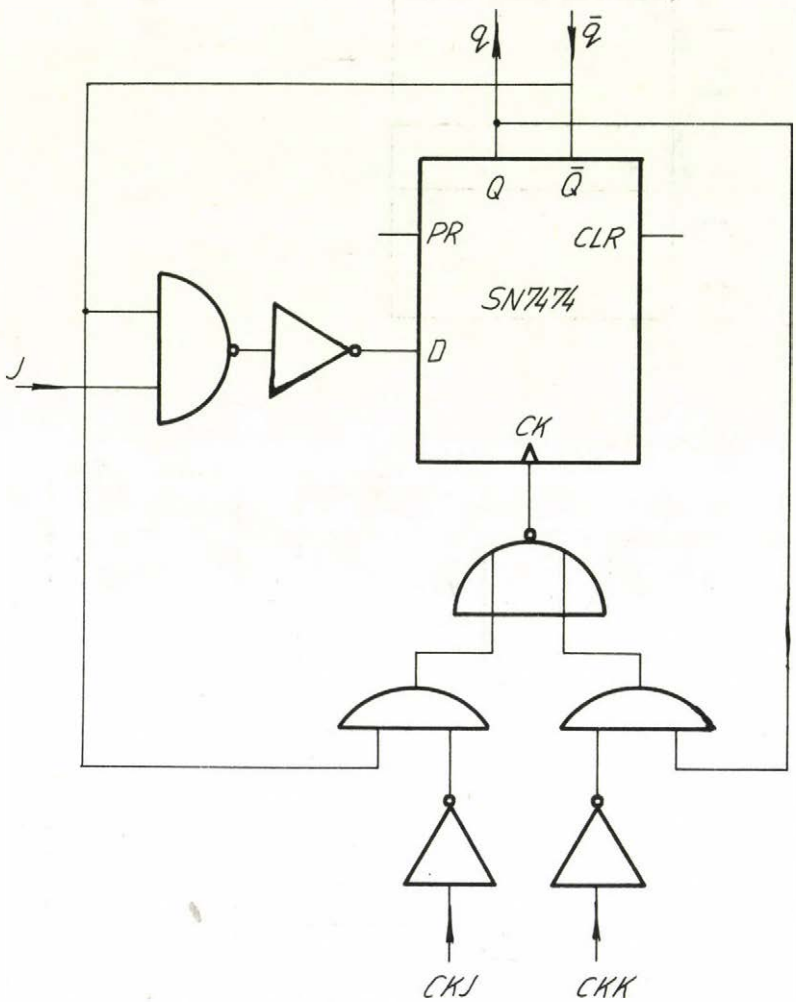


Fig. 5.

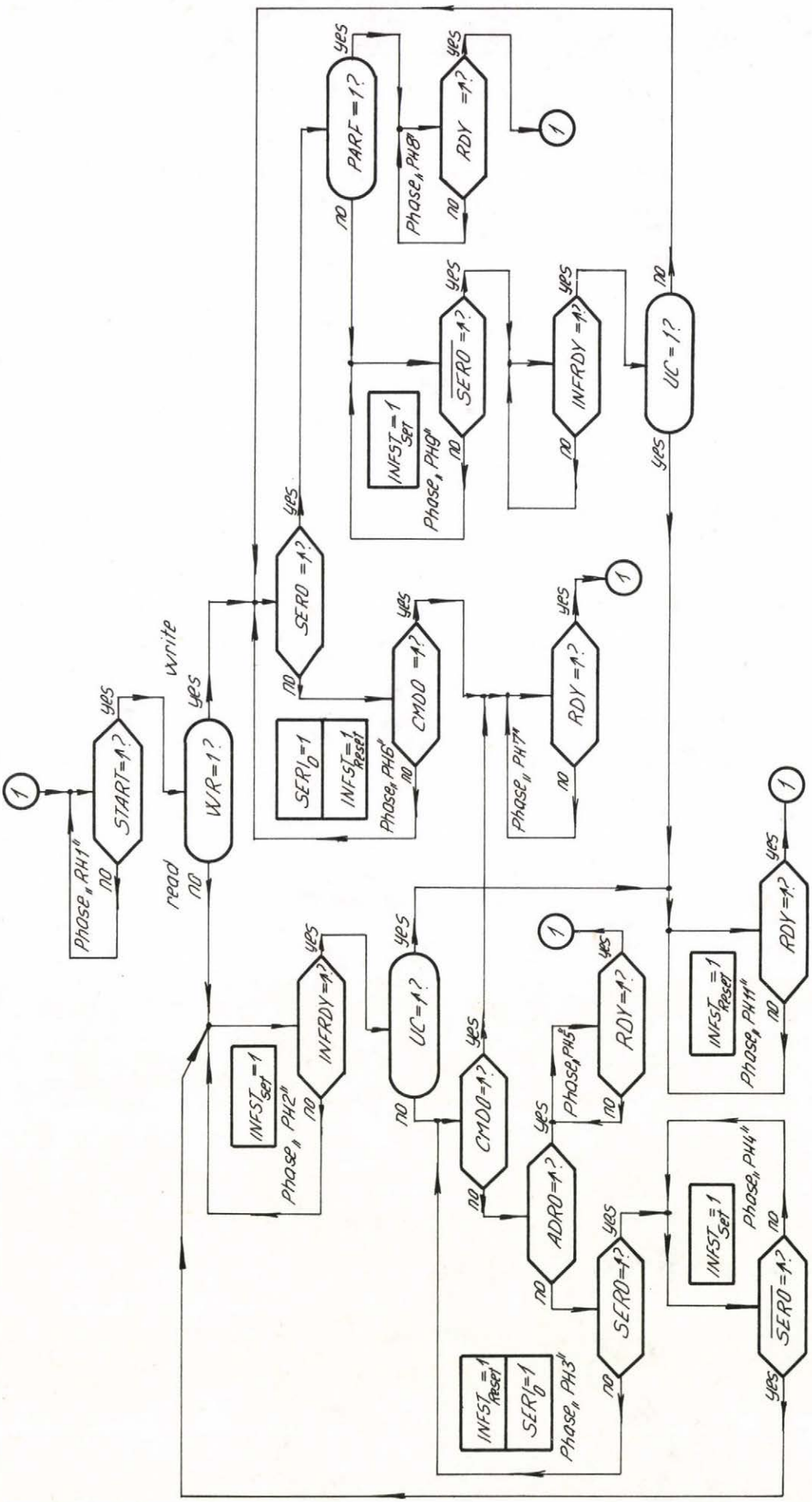

Phase „RH1"
START=1?
yes
no
write
read
WR=1?
INFST_Set=1
Phase „PH2"
INFRDY=1?
UC=1?
CMDO=1?
ADRO=1?
INFST_Reset=1
SERI_D=1
Phase „PH3"
SERO=1?
Phase „PH4"
RDY=1?
Phase „PH5"
Phase „PH6"
Phase „PH7"
Phase „PH8"
PARF=1?
Phase „PH9"
Phase „PH11"

Fig. 6

Phase	Phase reg. element	J	CKJ	CKK	Preset	Clear
PH1	1	PH5 = PH7+PH8+PH11	RDY	PH2 + PH6	ZER	„1"
PH2	1	PH1 · $\overline{WR}$	START	PH3 + PH11	„1"	$\overline{ZER}$
	2	PH4	$\overline{SERO}$		„1"	
PH3	1	PH2 · $\overline{UC}$	INFRDY	PH4+PH5+PH7	„1"	$\overline{ZER}$
PH4	1	PH3	SERO	PH2	„1"	$\overline{ZER}$
PH5	1	PH3	ADRO	PH1	„1"	$\overline{ZER}$
PH6	1	PH1 · WR	START	PH7+PH8+PH9	„1"	$\overline{ZER}$
	2	PH10 · $\overline{UC}$	INFRDY		„1"	$\overline{ZER}$
PH7	1	PH3 + PH6	CMDO	PH1	„1"	$\overline{ZER}$
PH8	1	PH6 · PARF	SERO	PH1	„1"	$\overline{ZER}$
PH9	1	PH6 · $\overline{PARF}$	SERO	PH10	„1"	$\overline{ZER}$
PH10	1	PH9	$\overline{SERO}$	PH6 + PH11	„1"	$\overline{ZER}$
PH11	1	(PH2+PH10) · UC	INFRDY	PH1	„1"	$\overline{ZER}$

a.)

$INFST_{Set} = PH2 + PH4 + PH9$ $\qquad$ $SER1 = PH3 + PH6$

$INFST_{Reset} = PH3 + PH6 + PH11$

b.)

Fig. 7.

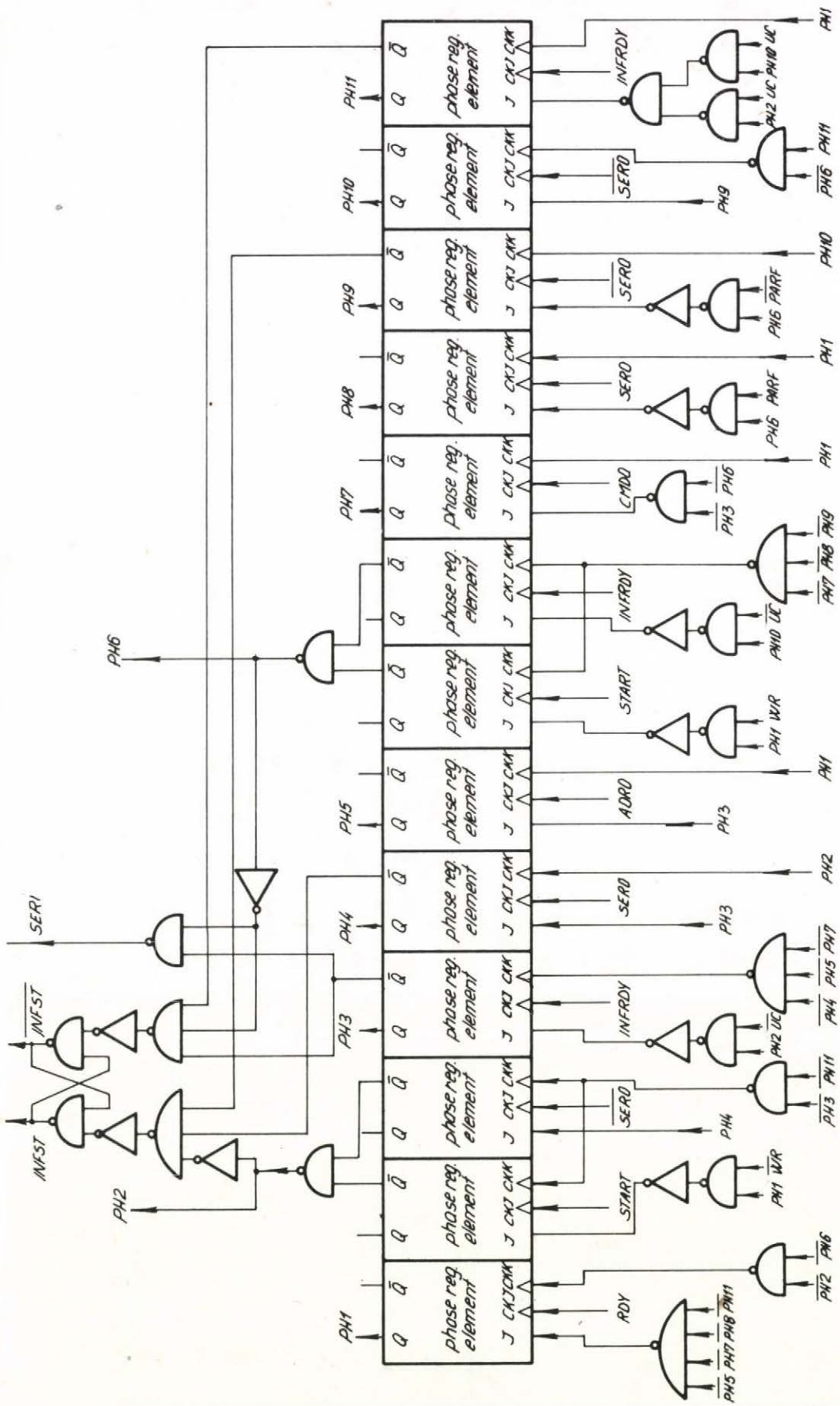

INFST
SERI
PH2
PH6
PH1
PH2
PH3
PH4
PH5
PH7
PH8
PH9
PH10
PH11
phase reg. element
J CKJ CKK
Q Q̄
START
SERO
ADRO
INFRDY
RDY
CMDO
PARF
WR

fig. 8

Э.А.ЯКУБАЙТИС, А.Ю.ГОБЗЕМИС, Г.Ф.ФРИЦНОВИЧ, В.П.ЧАПЕНКО

СИНТЕЗ И АНАЛИЗ ДИСКРЕТНЫХ УСТРОЙСТВ НА ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ МАТРИЦАХ

Институт электроники и вычислительной
техники АН Латвийской ССР, Рига, СССР

Введение.

Идея программируемой логики получила в настоящее время широкое развитие в направлении производства и применения программируемых логических матриц (ПЛМ), настраиваемых на реализацию заданного логического преобразования как в процессе изготовления, так и непосредственно пользователем [I,2] .

Рассмотрим реализацию дискретного устройства на одной ПЛМ с N входными, H промежуточными и L выходными шинами, функциональная блок-схема которой приведена на рис.I.

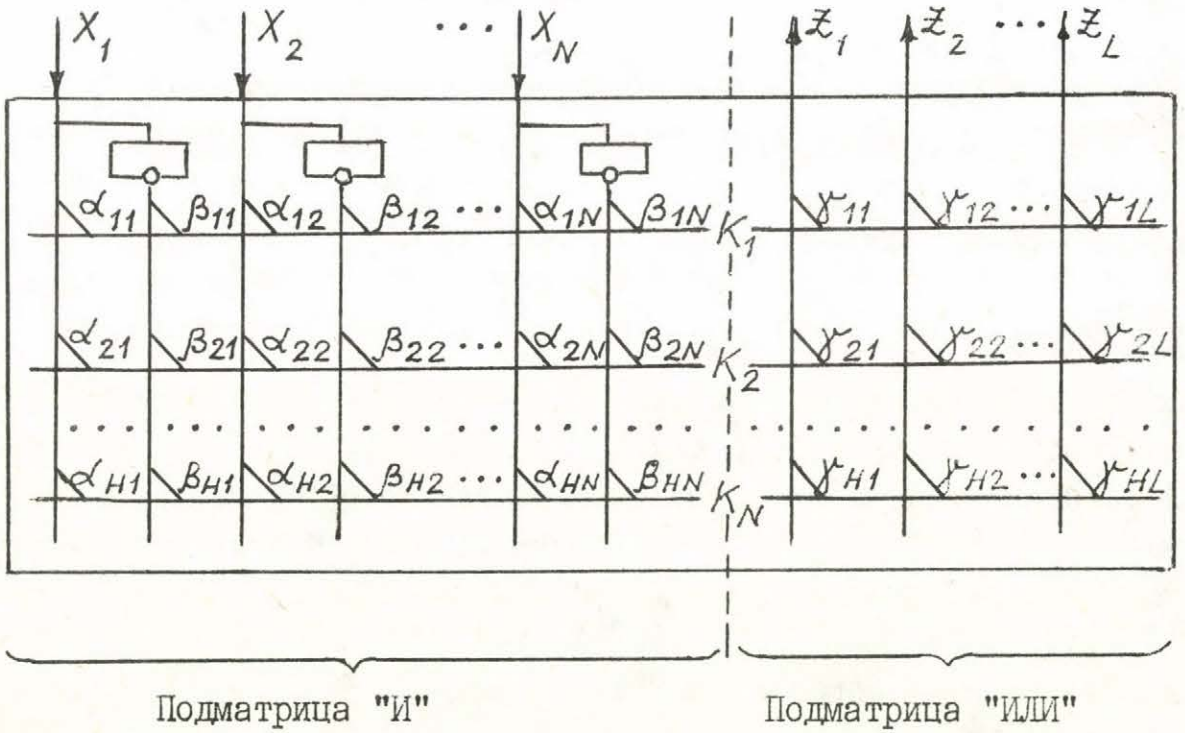


Рис.I. Функциональная модель ПЛМ.

Аналитической моделью ПЛМ может быть следующая система дизъюнктивных нормальных форм (ДНФ):

$$\left.\begin{aligned} K_j &= \bigwedge_{t=1}^{N} X_t \bar{X}_t \quad , \\ z_i &= \bigvee_{j=1}^{H} K_j \quad , \quad i = 1,2,\ldots,L \ . \end{aligned}\right\} \qquad (I)$$

Иными словами, можно считать, что до выполнения настройки на всех выходах ПЛМ реализуются функции $z_i \equiv 0$, являющиеся дизъюнкциями конъюнкций всех входных переменных X_t и их отрицаний, образуемых на промежуточных шинах. В таком случае настройку (программирование) ПЛМ можно интерпретировать, как преобразование системы уравнений (I), в некоторую другую систему ℓ ДНФ от h входных переменных, где $\ell \leq L$, $h \leq N$, такую, что для некоторых p и q $z_{i_p} \neq z_{i_q}$, $K_{j_p} \neq K_{j_q}$, $i_p, i_q \in \{1,2,\ldots \ell\}$, j_p, $j_q \in \{1,2,\ldots,h\}$, $h \leq H$.

Введенная модель может быть использована при изучении вопросов синтеза как комбинационных схем, так и схем с памятью. В последнем случае множество входных переменных $\{X_i\}$ включает в себя внешние и внутренние (промежуточные) переменные. В любом из этих случаев логическая структура синтезируемого устройства представляется в виде системы ДНФ, адэкватной комбинационной части ПЛМ. Поэтому в дальнейшем ограничимся рассмотрением именно такого представления логических структур.

I. Синтез логических структур

Зададим синтезируемый автомат $\mathcal{U}$ совокупностью $\langle \mathcal{R}, \mathcal{L}, \mathcal{K}, \varphi, f \rangle$ его состояний входов, выходов, внутренних состояний, функций переходов и выходов. При структурной реализации всем состояниям $\rho_\mu \in \mathcal{R}$, $\lambda_\nu \in \mathcal{L}$, $\varkappa_w \in \mathcal{K}$ соответствуют двоичные наборы входных, выходных и промежуточных сигналов: $(A_1^u, \ldots, A_n^u)$, $(z_1^v, \ldots, z_m^v)$, $(Y_1^w, \ldots, Y_k^w)$. Тогда $\{A_i^u\} \cup \{Y_j^w\} = \{X\}$ – множество входных переменных ПЛМ.

В настоящем разделе будем рассматривать задачи, возникающие при синтезе структур асинхронных автоматов на ПЛМ заданных размеров. Поэтому естественно в качестве одного из критериев

оптимальности выбрать площадной критерий. Под площадью структурной реализации автомата условимся понимать произведение числа различных членов в ДНФ системы функций, описывающих логический преобразователь автомата, на сумму указанных функций и их аргументов. Площадь ПЛМ при реализации автомата на логических элементах и естественных задержках можно оценить выражением $S = h(n+m+2k)$, где h - число различных членов в ДНФ системы функций переходов и выходов.

Логический преобразователь такого автомата может быть реализован в виде двухуровневой схемы на h $(n+k)$-входовых элементах И и $(m+k)$ h-входовых элементах ИЛИ. Другим критерием оптимальности структуры естественно считать её вес V, выражающийся в суммарном числе вхождений переменных и конъюнкций в уравнения системы. Таким образом, задача синтеза оптимальной структуры на ПЛМ сводится к минимизации функционала $\Phi = S \cdot V$. Синтез, направленный на достижение минимума Φ, объективно является сложной задачей. Один из возможных подходов к её практическому решению заключается в определении некоторой иерархии частных критериев оптимальности и в разработке методов решения задач структурного синтеза, направленных на оптимизацию по одному из этих критериев, выбранному в качестве основного. Оптимизация по подчиненным критериям осуществляется лишь в пределах найденного решения. Такими частными критериями могут быть:

- число k промежуточных переменных;
- число h членов в ДНФ системы функций ;
- число вхождений букв в конъюнкции и число конъюнкций в ДНФ системы.

Рассмотрим три направления синтеза логических структур с учетом частных критериев. Первое из них предполагает выбор в качестве основного критерия числа промежуточных переменных в системах функций переходов и выходов при кодировании внутренних состояний автомата Π-кодом. Число членов в ДНФ системы при этом играет второстепенную роль. Способ кодирования внут-

ренних состояний зависит от числа состояний и топологии связей между ними. Условия правильности Π -кода могут быть сформулированы следующим образом: если через $[x]$ обозначить кодовое слово, сопоставленное состоянию x , а через $\mathcal{J}_{i,j}$ – интервал, покрывающий $[x_i]$ и $[x_j]$, то код внутренних состояний допускает реализацию асинхронного автомата без гонок, если для каждой пары переходов между внутренними состояниями вида $(x_i \to x_j , x_\ell \to x_m)$, вызванных одним и тем же состоянием входов ρ (связанных пар переходов), справедливо

$$\mathcal{J}_{i,j} \cap \mathcal{J}_{\ell,m} = \emptyset \quad . \qquad (2)$$

Здесь предполагается, что $x_j \neq x_m$ и для всех $j \neq m$ $[x_j] \neq [x_m]$. Процедура кодирования внутренних состояний, основанная на условии (2), обычно выполняется после минимизации их числа. В результате может оказаться невозможным достижение абсолютного минимума промежуточных переменных. Известен ряд методов, совмещающих процедуры минимизации и кодирования состояний, которые основаны на том, что при выборе кода выполнение условия (2) требуется только для таких связанных пар переходов, в которых x_j и x_m несовместимы [3-5] . Пусть $\Pi^0 = \{\pi_1^0, \ldots, \pi_\alpha^0\}$ – множество связанных пар переходов автомата $\mathcal{U}$ таких, что для любого $\pi_i^0 = \{x_{i,\alpha} \to x_{i,\beta}; x_{i,\gamma} \to x_{i,\delta}\}$ состояние $x_{i,\beta}$ несовместимо с $x_{i,\delta}$ $(0 \leq i \leq \alpha)$. Пусть $\Pi = \{\pi_1, \ldots, \pi_K\}$ – покрытие множества Π^0 полными двухблочными разбиениями на множестве внутренних состояний. Рассмотрим пару совместимых внутренних состояний x_u и x_v , устойчивых при одном и том же состоянии входов ρ_w , для которых существует $\pi_j \in \Pi$ такой, что x_u и x_v не входят в один и тот же блок разбиения π_j . Множество всех пар переходов в такие состояния обозначим через Π' . Схема совместного решения задач минимизации и кодирования состояний асинхронного автомата тогда состоит в последовательном выполнении следующих действий: I) построение множества Π^0 ; 2) построение покрытия Π со свойством $\Pi \supseteq \Pi'$; 3) построение разбиения $P = \prod_{i=1}^{K} \pi_i$ $(\pi_i \in \Pi)$ и соответствующего ему автомата $\mathcal{U}'$ $(\mathcal{U}' \supseteq \mathcal{U})$. Покрытие Π , содержащее наименьшее число элементов, порождает Π -код минимальной длины. При последующей минимизации системы функций переходов и выходов могут быть учтены требования минимизации

числа h членов ДНФ системы, а также суммарного числа вхождений букв и членов в эту систему. Однако, стремление к поиску кода минимальной длины не всегда оправдано с точки зрения достижения оптимальной логической структуры автомата. Особенности реализации на ПЛМ часто выдвигают в качестве основного требования минимизацию числа членов ДНФ системы функций переходов и выходов. Число аргументов и функций может играть второстепенную роль. Рассмотрим подробнее основанный на этом предположении подход к структурному синтезу [6]. Пусть $R_0 = \{r_1, r_2, \ldots, r_\alpha\}$, $L_0 = \{l_1, l_2, \ldots, l_\beta\}$ - множества входных и выходных интервалов автомата $\mathcal{U}$ и пусть условия его работы заданы при помощи списка переходов. Переходы представим в виде: $(r_u, x_i \to x_j, l_v)$, где $r_u \in R_0$, $x_i, x_j \in \mathcal{K}$, $l_v \in L_0$. Такая запись означает, что переход из состояния x_i в состояние x_j происходит под воздействием любого входного набора из интервала r_u и в результате этого перехода на выходе автомата устанавливается некоторый выходной набор из интервала l_v. Рассмотрим произвольную пару переходов $P_1 = \{r_p, x_i \to x_j, l_v\}$ и $P_2 = \{r_q, x_g \to x_h, l_w\}$. Через $r_{p,q}$ обозначим минимальный интервал, покрывающий r_p и r_q, а через $\mathcal{K}_{p,q}$ - множество всех внутренних состояний автомата $\mathcal{U}$, в которые заданы переходы из состояний множества (x_i, x_j, x_g, x_h) под воздействием входных наборов интервала $r_{p,q}$. Переходы P_1 и P_2 назовем совместимыми, если и только если $\mathcal{K}_{p,q}$ является множеством совместимости, и выходные интервалы l_v и l_w содержат хотя бы один общий набор ($l_v \cap l_w \neq \emptyset$). Пусть S_t - некоторое множество попарно совместимых переходов автомата $\mathcal{U}$; R_t - минимальный интервал, покрывающий все входные интервалы, под воздействием которых происходят переходы, принадлежащие множеству S_t; $\mathcal{K}_{i_t}$ и $\mathcal{K}_{j_t}$ - множества внутренних состояний, из которых и в которые заданы эти переходы; $\mathcal{L}_t$ - минимальный интервал, покрывающий все выходные наборы, устанавливаемые в результате переходов из множества S_t. Множество $S_t = \{R_t, \mathcal{K}_{i_t} \to \mathcal{K}_{j_t}, \mathcal{L}_t\}$ будем называть S-переходом. Введенные обобщения позволяют сформулировать понятие противогоночного $\Pi(S,2)$-кода внутренних состояний автомата, свойством

которого является наличие хотя бы одной развязывающей переменной для каждой пары переходов вида $\{s_t, (r_q, x_g \to x_h, l_w)\}$, где $R_t \cap r_q \neq \phi$, $x_h \in K_{j_t}$.

Для формулировки условий минимизации числа членов в системе ДНФ рассмотрим некоторые дополнительные свойства S -переходов. S -переход вида $\{R_t, K_{i_t} \to K_{j_t}, L_t\}$ назовем устойчивым, если любой допустимый переход из внутренних состояний множества $K_{i_t} \cup K_{j_t}$ под воздействием любого из входных наборов множества R_t принадлежит данному S -переходу. Совокупность $S = \{s_1, s_2, \ldots, s_{p_s}\}$ устойчивых S -переходов называется полной, если $\bigcup_{t=1}^{p_s} s_t = \Sigma$, где Σ - множество всех переходов, заданных условиями работы автомата U. Совокупность S называется замкнутой, если для любых $s_t \in S$ и $r_q \in R_a$ существует $s_u \in S$ такое, что $s_t^q \subseteq s_u$. Здесь s_t^q -множество всех заданных переходов из любого состояния множества K_{j_t} под воздействием наборов из входного интервала r_q. Справедливо следующее утверждение [6]. Если автомат U имеет полную и замкнутую совокупность $S = \{s_1, s_2, \ldots, s_{p_s}\}$ устойчивых S -переходов, то кодирование его внутренних состояний $\Pi(S,2)$-кодом допускает представление в виде системы ДНФ с суммарным числом различных членов $p \leq p_s$.

Таким образом, процедура синтеза логической структуры автомата на ПЛМ сводится к последовательному выполнению следующих действий:

- построению минимальной по числу элементов совокупности S для заданного автомата U;
- преобразованию автомата U в автомат U' путем объединения состояний K_{j_t} любого $s_t \in S$ в одно внутреннее состояние и определению переходов между ними;
- построению $\Pi(S,2)$-кода внутренних состояний автомата U' и минимизации его длины;
- нахождению кратчайшей ДНФ, реализующей логическую структуру синтезируемого автомата на ПЛМ.

Укажем, наконец, на ещё один приближенный метод минимизации функционала φ , который обеспечивает минимальное число вхождений промежуточных переменных в конъюнктивные члены системы ДНФ. Он основан на методе инерционных подавтоматов [7] и может быть рекомендован, как сравнительно простой способ синтеза дискретных устройств на ПЛМ. Определим устойчивое полное состояние (у.п.с.) $\mathcal{M}_{i,h}$ автомата, как пару (x_i, ρ_h).

Пусть $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_h\}$ - множество у.п.с. автомата. Код внутренних состояний, вводимый в процессе формализации условий работы автомата в виде матрицы связей между у.п.с., позволяет каждое устойчивое полное состояние $\mathcal{M}_j$ однозначно представить конъюнкцией вида $\tilde{A}_{j_1}\tilde{A}_{j_2}\ldots\tilde{A}_{j_{n_j}}y_j$, где $\{A_{j_1}, A_{j_2}, \ldots, A_{j_n}\}$ - множество входных переменных, определенных в состоянии $\mathcal{M}_j$, y_j - промежуточная переменная. Функции переходов и выходов автомата могут быть представлены в виде следующей системы ДНФ:

$$\left.\begin{aligned} y_i &= \bigvee_{j=1}^{h} \varepsilon_{ij}\,\tilde{A}_{j_1}\ldots\tilde{A}_{j_{n_j}}\,y_j\,, \quad i=1,2,\ldots,h\,, \\ z_v &= \bigvee_{j=1}^{h} \nu_{v,j}\,\tilde{A}_{j_1}\ldots\tilde{A}_{j_{n_j}}\,y_j\,, \quad v=1,2,\ldots,m\,, \end{aligned}\right\} \quad (3)$$

где $\varepsilon_{i,j} = 1$, если задан переход из $\mathcal{M}_j$ в $\mathcal{M}_i$, и $\varepsilon_{ij} = 0$, если такой переход запрещен. Величина $\nu_{v,j}$ определяется значением v-й выходной переменной в j -ом у.п.с. Непосредственно по алгоритму функционирования мы можем построить структурную реализацию с площадью $S = h \times (n+m+2h)$. Минимизация величины S может быть достигнута эквивалентным преобразованием автомата $\mathcal{U}$ в $\mathcal{U}' \supseteq \mathcal{U}$ с числом обобщенных у.п.с. $h' \leq h$. Площадь структуры уменьшается, если $h' < h$. При этом число аргументов, функций и различных членов ДНФ системы (3) сокращается на одну и ту же величину.

2. Анализ логических структур

Для решения задач анализа преобразуем систему уравнений (1) к виду:

$$\left.\begin{aligned} K_j &= \bigwedge_{t=1}^{N} (\alpha_{t,j} \vee X_t)(\beta_{t,j} \vee \bar{X}_t) \quad , \\ Z_i &= \bigvee_{j=1}^{H} \gamma_{i,j} K_j \quad , \qquad i = 1,2,\ldots,L \quad . \end{aligned}\right\} \quad (4)$$

ПЛМ, как объект анализа, будем задавать парой $\langle \mathcal{U}, \Omega \rangle$, где $\mathcal{U}$ - реализуемый конечный автомат, а Ω - множество технических состояний матрицы, которому однозначно соответствует множество значений двоичных переменных $\{\alpha, \beta, \gamma\}$, введенных в (4). Эти переменные позволяют моделировать наличие либо отсутствие в конъюнкциях входных переменных $\{X_i\}$ в прямой либо в инверсной форме самих конъюнкций в ДНФ, а также содержание либо отсутствие уравнений в системе. Физический смысл множества $\{\alpha, \beta, \gamma\}$ становится ясным, если эти переменные поставить в соответствие плавким перемычкам ПЛМ (см. рис.I) и считать, что совокупность их значений кодирует внешнее воздействие на ПЛМ с целью установления её в заданное техническое состояние. Это воздействие осуществляет программирование матрицы на реализацию некоторой системы ДНФ. Естественно считать, что нулем кодируется отсутствие программирующего воздействия, а единицей - его наличие.

Непредвиденное изменение алгоритма функционирования синтезируемого устройства может быть интерпретировано как изменение технического состояния ПЛМ $\omega_i \in \Omega$ на $\omega_j \neq \omega_i$. Это изменение может быть вызвано либо ошибкой настройки, либо неисправностью некоторых компонент матрицы, возникшей в процессе её функционирования. Ограничив рассмотрение лишь логическими неисправностями, можно свести их к ошибкам настройки. Это позволяет с единых позиций сформулировать основные задачи анализа ПЛМ [8] . К ним относятся: во-первых, анализ возможных причин нарушения алгоритма функционирования ПЛМ и установление конкретной причины, вызвавшей данное нарушение, во-вторых, анализ возможности реализации конкретного преобразования $\mathcal{U}$ на ПЛМ с заданными размерами.

Пусть автомат $\mathcal{U}$ задан системой ДНФ функций $\{\Psi_i\}$ от переменных $\{x_1, x_2, \ldots, x_n\}$. Подставив $Z_i = \Psi_i \quad (i = 1,2,\ldots,L)$ в систему

уравнений (4), получим:

$$\Psi_i = \bigvee_{j=1}^{H} \bar{\gamma}_{i,j} \bigwedge_{t=1}^{N} (\alpha_{t,j} \vee X_t)(\beta_{t,j} \vee \bar{X}_t), \quad i=1,2,\ldots,L. \quad (5)$$

Систему уравнений (5) можно решать как относительно переменных множества $\{\alpha,\beta,\gamma\}$, так и относительно входных переменных $\{X_1, X_2, \ldots, X_N\}$. В первом случае определяем функциональные возможности ПЛМ, т.е. решаем задачу, при каких технических состояниях (настройках, неисправностях) ПЛМ реализует заданную систему функций $\{\Psi_i\}$. Во втором случае определяем множества входных наборов, позволяющие идентифицировать техническое состояние ПЛМ с заданной точностью.

Анализ функциональных возможностей ПЛМ позволяет решить задачу синтеза заданного логического преобразователя. Кроме того, если некоторое множество логических неисправностей не обнаруживается полным проверяющим тестом, то оно соответствует избыточным компонентам реализации устройства. Эта информация может быть использована для минимизации логической структуры. Наоборот, в процессе минимизации системы ДНФ может быть получена информация об избыточных буквах в конъюнкциях и самих конъюнктивных членах в уравнениях. Это может облегчить решение задачи идентификации технического состояния дискретного устройства.

3. Алгоритмическая база

Трудности, возникающие при решении задач синтеза и анализа, требуют разработки эффективных алгоритмов и последующей автоматизации основных этапов логического проектирования. Целесообразным представляется единый подход к решению указанных задач, основанный на раскраске вершин графов [9-II]. Суть подхода сводится к следующему. Рассмотрим конечный неориентированный граф $G(M, \tau)$, где M - множество его вершин. Любые две вершины $m_i, m_j \in M$ соединены ребром тогда и только тогда, когда $m_i \bar{\tau} m_j$. Здесь τ - бинарное отношение, которое, как и множество M, особым образом определяется для каждой конкретной задачи. Пусть $\mathcal{M}$ - множество внутренне устойчи-

вых подмножеств множества M. Очевидно, что все элементы множества $\mathcal{M}$ состоят из вершин графа G , попарно находящихся в отношении τ . Упомянутые ранее логико-комбинаторные задачи анализа и синтеза в конечном счете заключаются в поиске минимально-замкнутых покрытий множества M элементами множества $\mathcal{M}$. Это, в свою очередь, может быть сведено к нахождению минимальной по числу цветов, в заданном смысле правильной раскраски вершин графа G . Дополнительное условие, налагаемое на правильность раскраски, зависит от специфики решаемой задачи . Основное же преимущество этого подхода заключается в том,что искомое решение можно получить без предварительного нахождения множества $\mathcal{M}$, тем самым избежав необходимости образования и хранения больших массивов промежуточной информации.Проиллюстрируем сказанное на двух примерах.

3.I. При минимизации систем булевых функций в качестве множества M может быть принято множество всех её рабочих состояний. Бинарное отношение τ определим следующим образом. Для любых двух элементов $m_\alpha, m_\beta \in M$ $m_\alpha \tau m_\beta$ тогда и только тогда, когда для m_α и m_β существует рабочий интервал системы $\mathcal{Y}(m_\alpha, m_\beta)$. При раскраске используется дополнительное условие, называемое f-правильностью. Раскраска вершин графа $G(M,\tau)$ называется f-правильной тогда и только тогда,когда любым двум смежным вершинам графа сопоставлены различные цвета и любое подмножество одноцветных вершин покрывает рабочий интервал системы булевых функций. Поиск кратчайшей ДНФ заданной системы булевых функций может быть осуществлен по следующей схеме:

I) построить граф несклеиваемости $G(M,\tau)$;
2) найти минимальную по числу цветов f-правильную раскраску графа G :
3) для каждого множества одноцветных вершин графа построить максимальный рабочий интервал, покрывающий элементы этого множества.

3.2. При построении кратчайших проверяющих тестов комбинационных схем множеством M является совокупность неисправностей компонент логической схемы. Определим на этом множестве отношение τ . Будем считать, что $m_i \tau m_j$, если существует хотя бы один входной набор, различающий неисправности схемы $m_i, m_j \in M$. Правильную раскраску графа $G(M,\tau)$ назовем K-правильной тогда и только тогда, когда для всех элементов каждого из подмножеств, окрашенных в один цвет, существует входной набор, обнаруживающий любую из неисправностей схемы, которые соответствуют этому подмножеству. Тогда процедура вычисления кратчайших проверяющих тестов может быть построена на основе нахождения минимальной K-правильной раскраски графа G и соответствующего ей множества входных наборов.

С использованием изложенных подходов разработана система алгоритмов решения основных задач синтеза и анализа асинхронных конечных автоматов. Эти алгоритмы легли в основу разрабатываемой в настоящее время диалоговой системы автоматизации логического проектирования.

ЛИТЕРАТУРА

I. Якубайтис Э.А. Программируемый логический автомат. - Автоматика и вычислительная техника, 1975, № 5, с.1-6.
2. Рейлинг Дж. Программируемые логические матрицы - новый элемент систем обработки данных. - "Электроника", 1974, № 16, с.38-46.
3. Петренко А.Ф. Минимизация и кодирование внутренних состояний конечного автомата. - Труды третьего Международного семинара "Прикладные аспекты теории автоматов", Варна, 1975, т.1, с.101-109.
4. Ланге Э.Э., Фрицнович Г.Ф. О совместном решении задач минимизации и кодирования состояний асинхронного автомата - Труды II Международного симпозиума "Дискретные системы" ,

Дрезден, 1977, т.I, с.I46-I53.

5. Hallbauer G. Procedure for state reduction and assignment in one step in synthesis of asynchronous sequential circuits. - Proceedings of the Intern. Symp. on Discrete Systems, vol. 1, Riga, 1974, p. 272-282.
6. Якубайтис Э.А., Буль Е.С., Ланге Э.Э., Лемберский И.Г., Фрицнович Г.Ф., Чапенко В.П. Методика синтеза асинхронных автоматов на ПЛМ.- Автоматика и вычислительная техника , I980, № 4, с.23-3I.
7. Якубайтис Э.А., Гобземис А.Ю., Фрицнович Г.Ф. Синтез конечных автоматов методом инерционных подавтоматов. - Автоматика и вычислительная техника, I967, № 5, с.25-30.
8. Гобземис А.Ю., Скляревич А.Н. Перестраиваемая логическая структура, как объект технического диагностирования. - Труды пятого Международного семинара "Прикладные аспекты теории автоматов", Варна, I979, т.П, с.539-548.
9. Фрицнович Г.Ф. Об одном подходе к автоматизации синтеза асинхронных конечных автоматов. - Tanul. MTA SZTAKI, 21, Budapest, 1974, с. 85-100.
10. Фрицнович Г.Ф. Об использовании раскраски вершин графов при минимизации систем булевых функций. - Труды третьего Международного семинара "Прикладные аспекты теории автоматов", т.I, Варна, I975, с.I93-204.
11. Гобземис А.Ю., Дюр А.А., Фрицнович Г.Ф. Об одном подходе к вычислению кратчайших проверяющих тестов комбинационных логических схем. - Вычислительная техника. Материалы конференции по развитию технических наук в республике и использованию их результатов. Каунас, I976, с.5-8.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

K.SAPIECHA, K.AMBROZIAK, R.KOTT
M.NOWICKI, C.SZCZESNY, K.WALCZAK
INSTITUTE OF COMPUTER SCIENCE
WARSAW TECHNICAL UNIVERSITY

AUTOMATIC TEST PATTERN GENERATION SYSTEM

ABSTRACT

In the paper an automatic test pattern generation system worked out in the Institute of Computer Science of Warsaw Technical University is outlined.

1. INTRODUCTION

The rapid development of integrated circuit technology caused a wide interest in testing equipment and in test generation techniques. [1-9].

An Automatic Test Pattern Generation (ATPG) has been considered one of the most important problem in computer testing and diagnosing.

In the paper an ATPG system worked out in the Institute of Computer Science of Warsaw Technical University is outlined.

The system is a FORTRAN preprocessor and is written in FORTRAN. This concept seems to have numerous advantages. First of them is FORTRAN popularity. As most of engineers learn and practise FORTRAN programing, then it seems quite probable they would accept, such preprocessor more readily than any other specialized language. Moreover, FORTRAN seems to be the most implemented programing language of the family of computers and minicomputers.

The technique of preprocessing has provided fast and easy implementation of the system. Moreover it enables the users to utilize the FORTRAN itself with the whole of its debugging support. Preprocessor can easily be developed or modified.

2. THE PREPROCESSOR LANGUAGE

The preprocessor language consists of FORTRAN statements and preprocessor directives which are specialized for test-generation.

There are five classes of directives. They are the following:

1. editing directives,
2. memory menagement directives,
3. libraries menagement directives,
4. data base menagement directives,
5. test generation directives.

The preprocessor translates a source program into an object FORTRAN code.

Translation does not change the FORTRAN statements but it replaces the directives by their corresponding FORTRAN codes. Usually one CALL instruction replaces one directive and the directive parameters form a list of actual parameters of this instruction. The original text of directive remains still in an object code as a comment.

3. DATA STRUCTURES

The data on which the system procedures operate are standardized as much as possible. Since the data usually

describes a circuit structure, then these descriptions are seized in the standard frames. There are two such frames, each of them corresponding to one of the most utilized way of circit structure representation. First representation reflects the intermodule connections and the second one reflects intergate connections in a circuit.

The standard corresponding to the module level description is general and makes it possible to represent any circuit structure. The standard corresponding to the gate level description is the particular case of the above one. Each standard frame consists of six FORTRAN tables containing the required information. For example, in the first case they contain the information about the module types, the module to chip relations, the intermodule connections and the interpin connections.

The system contains numerous directives which operate on bases e.g.: a base dumping, erasing, extending, dividing, copying, deleting and so on.

It should be noted that for the base application only its name is needed.

4. DATA MANAGEMENT

The dynamic approach to the memory allocation has been provided as a fundamental part of the whole system. The dynamic allocation is maintained by the MEMAR system.

The MEMAR performs functions of three classes:

- dynamic allocation of tables and of sets of tables(bases),

- usage of libraries stored in magnetic disc files,
- usage of operation libraries located in the main memory.

The MEMAR is used by means of preprocessor directives. However, it is possible to use it directly calling the proper subroutines from the MEMAR source library.

The MEMAR maintains all the free space of program memory. The dynamic tables can be located in the memory on request. The expanding of the tables is possible up to the limits of the program memory. A table can be also delated from the memory and the space is retrieved in that case. The operations are executed in such a way that an optimization of the free memory space is provided.

The farther extension of data manipulation possibilities are the system libraries. There are two kinds of libraries: permanent libraries and operation ones. Their organization is most flexible so they can hold any sort of data admissible in the system. Usually they are used to store the bases containing the circuit structure descriptions, the tables containing the test sets or input or/and output sequence and so on.

As far as the permanent libraties are concerned the following operations are possible via MEMAR:

- to create a library and to extend it ,
- to fulfill it with the contents of indicated tables or bases ,
- to load library contents into dynamic tables or bases (load the table or basis from the library),

- to check whether the indicated logical element descriptions are stored in a permanent library.

Sometimes only a particular part of the permanent library, for example the fifth tables of bases, is used very often Then a library in the main memory can be created and loaded with this part only. This is called an operation library.

The most important feature of the library mechanism is that it provides a hierarchy approach to test-generation. Every circuit can be described by means of the modules contained in the library and then put into the library as the new module.

5. TEST-GENERATION STRATEGIES

As far as test-set-generation is concerned numerous procedures are supplied by a set of specialized system directives. It facilitates the choice of the best test-set-generation strategy for a given circuit and for a given class of failures. There are three main options which state the strategy. These are as follows:

1. a method of fault schedule filling,
2. a method of initial test-sequence working out,
3. a method of initial test-sequence completion.

In the system the two fault schedule filling ways are provided. First of them is full automated and it uses the check-points selection procedure. In the second method the fault schedule is worked by hand. There are three possibilities of the initial test-sequence preparation.

The initial test-sequence can be worked out by

a circuit designer or can be obtained by means of random input pattern generator or can be none.

By the system procedures numerous test-set-generation strategies can be formulated. The most recommended one is the following:

1. To fill a fault schedule by the use of checkpoints selection procedure.
2. To input an initial test sequence.
3. To simulate (although by deductive simulator) one by one every tests belonging to the initial test sequence. To remove the tested failures from the fault schedule after each single test simulation.
4. To take from the fault list a fault so far untested.
5. To generate (by test generator) a test for the fault taken.
6. To validate the test given and to find out the failures tested by this test.
7. To remove the failures mentioned above from the fault schedule.
8. To repeat the ponts 4-7 until the fault schedule is empty.

The initial test sequence is determined on the basis of the functional properties analysis of the considered circuit. A modification of Akers's functional test generation is applied so that to give the best sequence.

By the use of test generator a single test or a number of tests can be obtained. This generator works on the

principle of path parallel sensitization resulting from the extension of D - algebra. [11,12] .

For sequential circuits their combinational iteration models are constructed. If a circuit is an asynchronous one than validation of the test set is provided. It can be done by the use of deductive simulator, containing a normal simulator.

The system also provides a strategy for test generation if a class of failures in not "stuck-at" type. [10] .

6. CONCLUSIONS

The system has been developed on CYBER 70 computer since 1975. The exploitation of the system has proved validity of conception. The data manipulation and management mechanisms have highly improved system efficacy. The hierarchy of the system has made it possible to formulate optimal test-set-generation strategies for small as well as large circuits.

During the recent years numerous strategies for SSI and MSI chips have been evaluated on the system [13-15] . For combinational circuits the strategy proposed here seems to be satisfactory. For example the 37-elements test set for SN74181 chip has been easily obtained. In the case of sequential curcuits the application of the strategy is not always successful. It results from the weakness of test-generator in case of the application of iterative models of sequential circuits. For this reason the

functional test generation is a very important point of the strategy and if well done the test-set-generation process can be highly improved.

As it was mentioned before the system hierarchy has made it possible to formulate test-set-generation strategies for large circuits and particularly for printed boards. That is the aim of our present investigations.

References

[1] Akers S.B.: A logic system for fault test generation. IEEE Trans.Comp.,June 1976.

[2] Bennets R.G., Brittle D.C., Prior A.C., Washington L.L. A modular approach to test sequence generation for large digital networks, Digital Processes,vol.1,1975, No.1.

[3] Bouricius W.G., Hsich E.P., Putzolu G.R., Roth J.P., Schneider P.R., Tan Ch.I.: Algorithms for detection of faults in logic circuits, IEEE Trans.Comput.,vol. C-20, No.11, Nov.1971.

[4] Breuer M.A.: A random and an algorithmic technique for fault detection test generation for sequential circuits IEEE Trans. Comput.,vol.C-20, November 1971.

[5] Breuer M.A., and Harrison L.: Procedures for eliminating static and dynamic hazards in test generation, IEEE Trans.Comp.Oct.1974.

[6] Breuer M.A.: Survey of digital logic simulators and automatic test generation systems. Breuer Associates, 1974.

[7] Breuer M.A., Friedman A.D.: Diagnosis;reliable design of digital systems, Computer Science Press, Inc.,1976.

[8] Muth P.: A nine-valued circuits model for test generation. IEEE Trans.Comp.,June 1976.

[9] Putzolu G.R., Roth J.P.: A heuristic algorithm for the testing ~~of the testing~~ of asynchronous circuits, IEEE Trans.Comput., vol.C-20, June 1971.

[10] Nowicki M: A diagnostic models for any-type fault test generation in switching circuits. Digital Processes, No. 3/4, 1979.

[11] Sapiecha K., Nowicki M.: Algorytm generacji testów diagnostycznych metodą równoległego uczulania ścieżek. Międzynarodowe Sympozjum FTC, Wisła 1976.

[12] Sapiecha K.: A path set sensitization procedure for generating tests for combinational circuits. Digital Processes, No.4, 1976.

[13] Sapiecha K.; Philosophy of design- verification and test-generation system, Proc. of III Int.Conf. FTSD79, Brno, 1979, pp.154-165.

[14] Sapiecha K., Walczak K., Nowicki M: Test set generation, submitted for publication in MTA SZTAKI TANULMANYOK.

[15] Sapiecha K., Szczęsny C., Walczak K.: Evaluation of test-set-generation strategies, submitted for FTSD80.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

ZDZISŁAW B. MIADOWICZ

ON THE PROBLEM OF THE CONNECTEDNESS OF THE PERIODIC SUM OF FINITE AUTOMATA

Institute of Control Engineering
Technical University of Poznan

INTRODUCTION

In this paper the connectedness of the periodic sum of finite automata is discussed. The periodic sum of finite automata has been studied in [1-6]. The connectedness of the periodic sum of finite automata has been discussed in all of these papers. The main attention has been paid in [5,6] to the connectedness of the periodic sum of the strongly connected asynchronous automata and strongly connected permutation automata. The conditions of the strong connectedness of the periodic sum of automata have been found in these papers was such, that we must known the structure of the fixed analog of the periodic sum of automata to say that it is strongly connected or not. In this paper some conditions for solution the problem of the connectedness of the periodic sum of finite automata are given only on the ground of the knowledge of the structures properties of the automata.

1. PRELIMINARY REMARKS

In this section we give the basic definitions to be used later. An automaton A is a triple (S, Σ, M) where S is a nonempty finite state set, Σ is a nonempty finite input set, and $M: S \times \Sigma \longrightarrow S$ is the next state function /or the transition function/.
I is the free semigroup of strings over the alphabet Σ with the operation of string concatenation. We assume that I contains the empty string e.
Let i be a natural number. Then for i we define nonempty set Σ^i as:

$$\Sigma^i = \{ \sigma_1 \sigma_2 .. \sigma_i \; ; \; \sigma_1, \sigma_2, \ldots, \sigma_i \text{ in } \Sigma \} .$$

We extend the next state function M to a function of $S \times I$ into S defining recursively

$$M(s,e) = s \text{ for all } s \text{ in } S,$$

$$M(s,x\sigma) = M(M(s,x),\sigma) \text{ for all } x \text{ in } I \text{ and } \sigma \text{ in } \Sigma .$$

An automaton $A = (S, \Sigma, M)$ is strongly connected if and only if given any s_1, s_2 in S, there exists x in I such that $M(s_1,x) = s_2$.

The strictly periodic automaton or briefly periodic automaton V is a triple (S^+, Σ, M^+), where S^+ is a sequence $S_0^+, S_1^+, \ldots, S_{r-1}^+$ of finite nonempty state sets, Σ is a finite nonempty input set, M^+ is a sequence $M_0^+, M_1^+, \ldots, M_{r-1}^+$ of the next state functions, where

$$M_i^+ : S_i^+ \times \Sigma \longrightarrow S_{i+1/\mathrm{mod}\ r/}^+$$

for i in $\{0,1,\ldots,r-1\}$.

The number r is said to be a period of V.

Let $A^0 = (S^0, \Sigma, M^0)$, $A^1 = (S^1, \Sigma, M^1)$, ..., $A^{r-1} = (S^{r-1}, \Sigma, M^{r-1})$ be automata, let $\Psi_0 : S^0 \longrightarrow S^1$, $\Psi_1 : S^1 \longrightarrow S^2$, ..., $\Psi_{r-1} : S^{r-1} \longrightarrow S^0$ be one-to-one and onto functions. Then the periodic sum of automata $A^0, A^1, \ldots, A^{r-1}$ associated with functions $\Psi_0, \Psi_1, \ldots, \Psi_{r-1}$ is a periodic automaton $V = (S^+, \Sigma, M^+)$ with period r, where S^+ is a sequence $S^0, S^1, \ldots, S^{r-1}$ and M^+ is a sequence $M_0^+, M_1^+, \ldots, M_{r-1}^+$, where for each s in S^i, σ in Σ, and i = 0,1,..., r-1 we have

$$M_i^+(s,\sigma) = \Psi_i M^i(s,\sigma).$$

A fixed analog V^* of the periodic automaton $V = (S^+, \Sigma, M^+)$ is an automaton (S^*, Σ, M^*) where $S^* = S_0^+ \cup S_1^+ \cup \ldots \cup S_{r-1}^+$ and $M^* : S^* \times \Sigma \longrightarrow S^*$ is the next state function defined for each s in S_i^+, σ in Σ, and i = 0,1,...,r-1 as follows

$$M^*(s,\sigma) = M_i^+(s,\sigma).$$

2. CONNECTEDNESS OF THE PERIODIC SUM OF FINITE AUTOMATA

Theorem 1.

Let $A^0 = (S^0, \Sigma, M^0)$, $A^1 = (S^1, \Sigma, M^1), \ldots, A^{r-1} = (S^{r-1}, \Sigma, M^{r-1})$ be automata such, that for each proper subset T^i of the set S^i where $i = 0,1,\ldots,r-1$, we have

$$|\{M^i(T^i, \Sigma)\}| > |T^i| .$$

Let $\psi_0 : S^0 \to S^1$, $\psi_1 : S^1 \to S^2, \ldots, \psi_{r-1} : S^{r-1} \to S^0$ be one-to-one and onto functions, then the fixed analog V^* of the periodic sum V of the automata A^0, A^1, ..., A^{r-1} associated with functions ψ_0, ψ_1, ..., ψ_{r-1} is strongly connected.

Proof.

Assume for the proof, that the fixed analog V^* is not strongly connected. Let T^0, $T^1, \ldots$, T^{r-1} be proper subsets of the sets S^0, $S^1, \ldots$, S^{r-1}, respectively, and such that according to the [2,3]

$$\{\psi_i(s) : s \text{ in } \{M^i(T^i, \Sigma)\}\} \subseteq T^{i+1/\text{mod } r/}$$

where $i = 0,1,\ldots,r-1$.

It means that

$$|\{\psi_i(s) : s \text{ in } \{M^i(T^i, \Sigma)\}\}| \leq |T^{i+1/\text{mod } r/}|$$

From the assumption we have that for all automata

$$|\{\psi_i(s) : s \text{ in } \{M^i(T^i, \Sigma)\}\}| > |T^i|$$

and because it holds for the arbitrary proper subset T^i of the set S^i then we can say

$$|T^{i+1/\text{mod } r/}| > |T^i| .$$

It means, that

$$|T^1| > |T^0| ,$$
$$|T^2| > |T^1| ,$$
$$\vdots$$
$$|T^{j+1}| > |T^j| ,$$
$$\vdots$$
$$|T^0| > |T^{r-1}| .$$

So we get the contradiction. QED.

Lemma 1.
Let $A = (S, \Sigma, M)$ be an automaton such, that for each state s in S there exists σ in Σ such, that $M(s,\sigma) = s$ then for the arbitrary proper subset T of the set S we have

$$|\{M(T,\Sigma)\}| \geq |T| .$$

Proof.
The proof will be ommited.

Lemma 2.
Let $A = (S, \Sigma, M)$ be a strongly connected automaton and such, that for each state s in S there exists σ in Σ such that $M(s,\sigma) = s$ then for the arbitrary proper subset T of the set S we have

$$|\{M(T,\Sigma)\}| > |T| .$$

Proof.
The proof is obvious.

We obtain on the ground of Theorem 1, Lemmal and Lemma 2

Corollary 1.
Let $A^0 = (S^0, \Sigma, M^0)$, $A^1 = (S^1, \Sigma, M^1), \ldots, A^{r-1} = (S^{r-1}, \Sigma, M^{r-1})$ be automata such, that for each state s in S^i, where i = 0,1, ...,r-1 there exists σ in Σ such that $M^i(s,\sigma) = s$. Let from among the automata A^0, $A^1, \ldots, A^{r-1}$ at least one be the strongly connected automaton. Let $\psi_0 : S^0 \to S^1$, $\psi_1 : S^1 \to S^2, \ldots,$ $\psi_{r-1} : S^{r-1} \to S^0$ be one-to-one and onto functions, let $V = (S^+, \Sigma, M^+)$ be the periodic sum of the automata A^0, $A^1, \ldots, A^{r-1}$ associated with functions $\psi_0, \psi_1, \ldots, \psi_{r-1}$. Then the fixed analog V^* of the periodic sum V is strongly connected.

Theorem 2.
Let $A^0 = (S^0, \Sigma, M^0)$, $A^1 = (S^1, \Sigma, M^1), \ldots, A^{r-1} = (S^{r-1}, \Sigma, M^{r-1})$ be automata, let $\psi_0 : S^0 \to S^1$, $\psi_1 : S^1 \to S^2, \ldots, \psi_{r-1} : S^{r-1} \to S^0$ be one-to-one and onto functions. The fixed analog V^* of the periodic sum $V = (S^+, \Sigma, M^+)$ of the automata $A^0, A^1, \ldots, A^{r-1}$ associated with functions $\psi_0, \psi_1, \ldots, \psi_{r-1}$ is strongly connected if and only if for the arbitrary proper subset T^i of the set S^i, i = 0,1,...,r-1 we have

$$\text{(i)} \quad |\{M^*(T^i, \Sigma^r)\}| > |T^i|$$

or

$$\text{(ii)} \quad |\{M^*(T^i, \Sigma^r)\}| \leq |T^i| \;\&\; \{M^*(T^i, \Sigma^r)\} \not\subseteq T^i$$

Proof.

Necessary condition. Let V^* be strongly connected. Consequently each state s in S^i must be reached from the arbitrary proper subset T^i of the set S^i and it make good both conditions.

Sufficient condition. Denote the arbitrary proper subset of the set S^i by T_1^i and keep on doing $T_j^i = \{M^*(T_{j-1}^i, \Sigma^r)\}$ where $j = 2,3,\ldots$. From the conditions (i) , (ii) we obtain at once that for each i in $\{0,1,\ldots,r-1\}$

$$T_1^i \cup \{\bigcup_{j=2,3,\ldots} T_j^i\} = S^i.$$

So it means that V^* is strongly connected. QED.

Theorem 3.

Let $A^0 = (S^0, \Sigma, M^0)$, $A^1 = (S^1, \Sigma, M^1), \ldots, A^{r-1} = (S^{r-1}, \Sigma, M^{r-1})$ be strongly connected automata, let $\Psi_0 : S^0 \to S^1$, $\Psi_1 : S^1 \to S^2$, $\ldots$, $\Psi_{r-1} : S^{r-1} \to S^0$ be one-to-one and onto functions. The fixed analog V^* of the periodic sum $V = (S^+, \Sigma, M^+)$ of the automata $A^0, A^1, \ldots, A^{r-1}$ associated with functions $\Psi_0, \Psi_1, \ldots, \Psi_{r-1}$ is strongly connected if for the arbitrary proper subset T^i of the set S^i of the automaton A^i, where i in $\{0,1,\ldots,r-1\}$ we have

$$\text{(i)} \quad |\{M^*(T^i, \Sigma^r)\}| > |T^i|$$

or

$$\text{(ii)} \quad |\{M^*(T^i, \Sigma^r)\}| \leq |T^i| \;\&\; \{M^*(T^i, \Sigma^r)\} \not\subseteq T^i.$$

Proof.

Denote the arbitrary proper subset of the set S^i by T_1^i and keep on doing $T_j^i = \{M^*(T_{j-1}^i, \Sigma^r)\}$ where $j = 2,3,\ldots$. It means from the conditions (i) , (ii) that

$$T_1^i \cup \{\bigcup_{j=2,3,\ldots} T_j^i\} = S^i$$

and because the automaton A^i is strongly connected then

$$\{M^*(S^i, \Sigma^r)\} = S^i$$

hence V^* is strongly connected since the function ψ_i is one--to-one and onto and all automata are strongly connected. QED.

Theorem 4.
Let $A^0 = (S^0, \Sigma, M^0)$, $A^1 = (S^1, \Sigma, M^1), \ldots, A^{r-1} = (S^{r-1}, \Sigma, M^{r-1})$ be automata, let $\psi_0 : S^0 \to S^1$, $\psi_1 : S^1 \to S^2, \ldots$, $\psi_{r-1} : S^{r-1} \to S^0$ be one-to-one and onto functions. If there exists in arbitrary automaton A^i in the set S^i the state s such that s is not in $\{M^i(S^i, \Sigma)\}$ then the fixed analog $V^* = (S^*, \Sigma, M^*)$ of the periodic sum V of the automata $A^0, A^1, \ldots, A^{r-1}$ associated with functions $\psi_0, \psi_1, \ldots, \psi_{r-1}$ is not strongly connected.

Proof.
Let there exists s in S^i and s does not belong to the set $\{M^i(S^i, \Sigma)\}$, i in $\{0,1,\ldots,r-1\}$ and denote $t = \psi_i(s)$. It is easy to note, that the state t does not belong to the set $\{M^*(S^*, \Sigma)\}$ and consequently V^* is not strongly connected.

QED.

Theorem 5.
Let $A^0 = (S^0, \Sigma, M^0)$, $A^1 = (S^1, \Sigma, M^1), \ldots, A^{r-1} = (S^{r-1}, \Sigma, M^{r-1})$ be automata such, that there exists for each state s in S^i, $i = 0,1,\ldots,r-1$ the state t in S^i and σ in Σ such, that $M^i(t, \sigma) = s$. Let $\psi_0 : S^0 \to S^1$, $\psi_1 : S^1 \to S^2, \ldots$, $\psi_{r-1} : S^{r-1} \to S^0$ be one-to-one and onto functions. Let $A^{i_1}, A^{i_2}, \ldots, A^{i_q}$ be all automata in the set $\{A^0, A^1, \ldots, A^{r-1}\}$ such that for each $T^{i_m} \subset S^{i_m}$, $m = 1,2,\ldots,q$ we have $|\{M^{i_m}(T^{i_m}, \Sigma)\}| > |T^{i_m}|$. Let $A^{i_{q+1}}, A^{i_{q+2}}, \ldots, A^{i_r}$ be the set of all other automata from among the automata $A^0, A^1, \ldots, A^{r-1}$. Let be fixed the number p_j for each from the automata A^{i_j}, $j = q+1, q+2, \ldots, r$ such that for arbitrary proper subset T^{i_j} of the set S^{i_j} we have

$$|T^{i_j}| - |\{M^{i_j}(T^{i_j}, \Sigma)\}| \leq p_j .$$

The fixed analog V^* of the periodic sum $V = (S^+, \Sigma, M^+)$ of the automata A^0, A^1,..., A^{r-1} associated with functions ψ_0, ψ_1, ..., ψ_{r-1} is strongly connected if the inequality holds

$$q > \sum_{j=q+1}^{r} p_j .$$

Proof.

Let us assume that in each automaton A^i, $i = 1,2,\ldots,r-1$ we have for each proper subset T^i of the set S^i that

$$|\{M^i(T^i, \Sigma)\}| > |T^i| .$$

So, the number $q = r-1$. Let for the automaton A^0 the fixed number p_0 be positive and $p_0 = k$. We make a choise of the set S^0 the subset T^0 such that

$$|T^0| - |\{M^0(T^0, \Sigma)\}| = k .$$

Let $|T^0| = a$ then $|\{M^0(T^0, \Sigma)\}| = a-k$ and furthermore

$$|\{\psi_0(s) : s \text{ in } \{M^0(T^0, \Sigma)\}\} = T^1| = a-k$$

because ψ_0 is function one-to-one and onto.

Since for each proper subset T^1 of the set S^1 we have

$|\{M^1(T^1, \Sigma)\}| > |T^1|$ then according to the Lemma 2

$$|\{M^1(T^1, \Sigma)\}| \geq a-k+1$$

otherwice

$$|\{\psi_1(s) : s \text{ in } \{M^1(T^1, \Sigma)\}\} = T^2| \geq a-k+1$$

and furthermore

$$|\{M^2(T^2, \Sigma)\}| \geq a-k+2$$

and so on and in the end

$$|\{M^{r-1}(T^{r-1}, \Sigma)\}| \geq a-k+r-1$$

or

$$|\{M^{r-1}(T^{r-1}, \Sigma)\}| \geq a-k+q .$$

We have from [2,3] that the fixed analog V^* of the periodic

sum V is not strongly connected if

$$\{\Psi_{r-1}(s) : s \text{ in } \{M^{r-1}(T^{r-1}, \Sigma)\}\} \leq T^0$$

or

$$|\{\Psi_{r-1}(s) : s \text{ in } \{M^{r-1}(T^{r-1}, \Sigma)\}\}| \leq |T^0|$$

and furthermore

$$a-k+q \leq a$$

and finally

$$q \leq k$$

so we get a contradiction. QED.

REFERENCES

[1]. Gecseg F., Imreh B. - On the periodic sum of finite automata. Recive to print in Foundations of Control Eng.

[2]. Grzymała-Busse J. - On the connectivity of the periodic sum of automata. Proc. of the Symp. and Summer School Math. Found. of Comp. Sci. High Tatras, Sep 3-8 /1973/ 231-235.

[3]. Miądowicz Z., Mikołajczak B. - On the connectivity of the periodic sum of finite automata. Found. of Contr. Eng. 1, 2 /1976/, 97-102.

[4]. Miądowicz Z. - Some problems concerning with the periodic sum of finite automata. Proc. of 2-nd IFAC Symp. on "Discrete Systems" Dresden Vol 5 /1977/, 94-103.

[5]. Miądowicz Z. - Some structural properties of the periodic sum of finite automata. Found. of Contr. Eng. 3,2/1978/

[6]. Miądowicz Z. - Strongly Connectedness of the periodic sums of finite automata./in russish/ Tanulmanyok MTA SzTAKI 99 /1980/ 54-61.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

L. HARMAT

САМОПРОВЕРКА В МУЛЬТИПРОЦЕССОРНЫХ СТРУКТУРАХ

Институт по Координации Вычислительной Техники
Будапешт

СОДЕРЖАНИЕ

Потребность в контроле и диагностике мультипроцессорных машин. Возможность самопроверки и самодиагностики.

Краткий обзор работ, проведенных в области теории самодиагностики, системодиагностики мультипроцессорных систем, на основе технической литературе.

Представление новой диагностической модели, описывающей структуру и тесты мультипроцессорных систем.

Модель отражает различные черты структуры и тестов с подробностью, соответствующей практике современных микропроцессорных систем.

Введение понятия детектируемости (обнаруживаемости) системных ошибок и установление необходимых и достаточных условий данной степени детектируемости.

Постановка задач по анализу детектируемости структур и наборов тестов, а также по синтезу (спецификации) набора тестов для достижения данной детектируемости.

1. ВВЕДЕНИЕ

Использователи вычислительных машин желают решать задачи надежно и без остановок. Из множества условий надежной и непрерывной работы выделяем надежность аппаратной части.

Из причинного соотношения между состояниями – ошибочными или нормальными – в следующие друг за другом моменты времени вытекает необходимость в периодической проверке состояний машины.

В случае ошибочного состояния (вследствие ошибки аппаратной части) необходимо иметь эффективную диагностику, то есть локализацию, а потом обмен или ремонт ошибочного блока. Целесообрано иметь диагностику на уровне наименьшего элемента замены.

При современном уровне вычислительной техники, который характеризуется микропроцессорами и другими схемами БИС, появился и все распространяется новый тип устройств: многомикромашинные комплексы, или другими словами, мультимикропроцессорные машины.

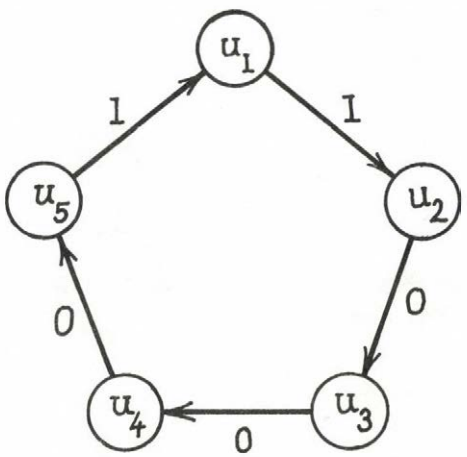
В этом типе устройств и по своей микропроцессорной структуре, и по своей конструктивной однородности легко осуществить самопроверку и самодиагностику.

Это значит, что процессорные модули могут проверить друг друга и остальные конструктивные модули, а также на основе таких тестов можно провести диагностику.

Исследования самопроверяющих и самодиагностических мультипроцессорных машин потребует создания соответствующей диагностической модели.

2. НОВАЯ ДИАГНОСТИЧЕСКАЯ МОДЕЛЬ

В технической литературе известно несколько моделей самодиагностических структур и их тестов, например: [1] ÷ [4] .
Обзоры и сравнения моделей находятся в [5] ÷ [8] .
На рис. 1. и на рис. 2 показаны примеры изображения структур в моделях по [1] и [2].



вершины: узлы, способны самостоятельно проверить другие узлы;

дуга от u_i к u_j : связь для теста (u_j проверяется узлом u_i)

пометки дуги $u_i \rightarrow u_j$

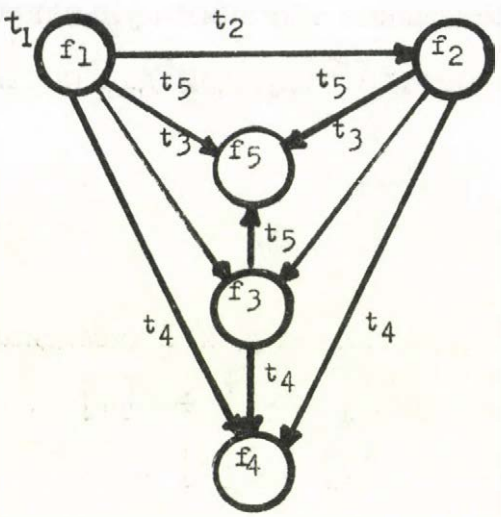
0 - оценка "в порядке" для u_j;

1 - оценка "ошибка" для u_i.

Модель примерной структуры по литературе [1]

Рис. 1.

Необходимость в создании новой диагностической модели следует из того, что эти модели или слишком упростили описание тестовых процессов, не отражали роли отдельных и содействующих структурных узлов, или были сформулированы в терминах ошибок,



вершины: ошибки;

дуга от f_i к f_j помеченная t_k: существует тест t_k, проверяющий ошибку f_j, результат которого неверный в присутствии ошибки f_i;

пометка t_1 вершины f_i : существует тест t_1 для f_i (результат всегда верный)

Модель примерной структуры по литературе [2]

Рис.2.

а не в конструкционных терминах, что не разрешает использовать конструкционно-структурную однородность микромашин. Мало внимания было обращено на описание связей, как конструктивных элементов.

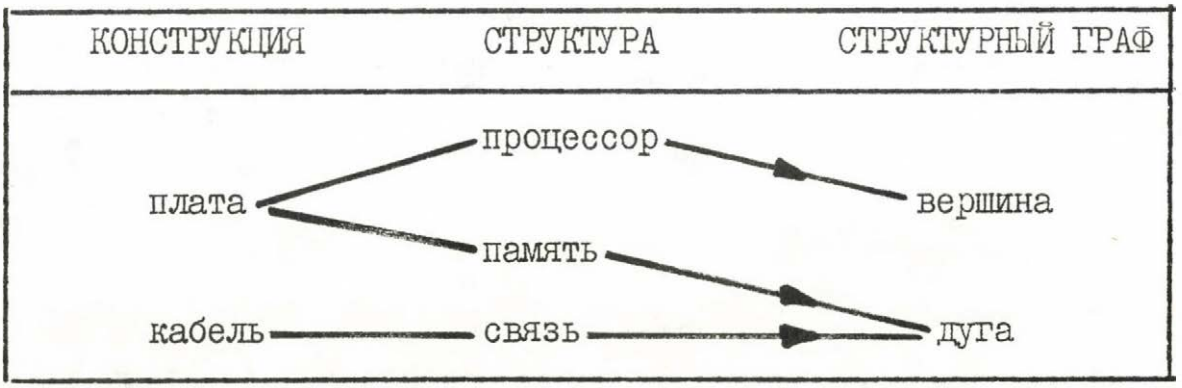
2.I. Описание структуры

Неформально структура может быть определена как "вид об аппаратной части машины глазами разработчика логики". Структура S состоит из структурных узлов U и из связей L между ними. Структуру можно представить с разной подробностью. Для целей проверки и диагностики выбирается то структурное разложение, при котором структурные элементы в наибольшей мере совпадают с наименьшими конструктивными элементами замены. Типичные такие элементы – одноплатная микромашина (single board microcomputer), интеллигентное устройство управления вводом-

выводом, память. При этом структурным связям соответствуют такие конструктивные элементы как кабели, панели или участки панелей.

Структура $S=(U, L)$ описывается ориентированным графом $G=(N, A)$, где N – множество вершин и A – множество дуг. При этом каждому структурному элементу, который имеет способность самостоятельной обработки, соответствует вершина графа. Внешние устройства могут быть изображены также вершинами. Оперативную и постоянную память считаем как "связи с переменным временем задержки". Этим виртуальным связям как и обычным связям соответствуют дуги графа. В случае связей данных дуга ориентируется по направлению потока данных, а в случае связей управления от управляющей вершины к управленной.

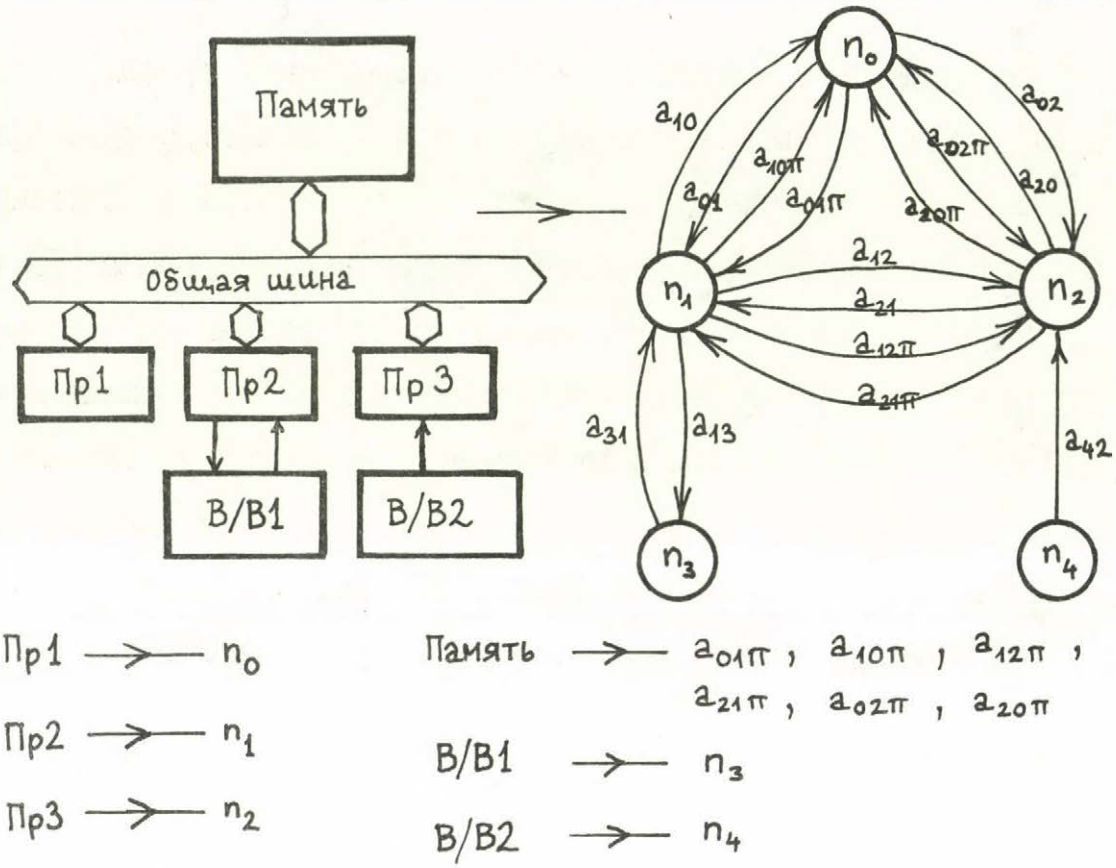
Произведение структурного графа и отношение его элементов к элементам конструкции показаны на рис. 3.



Соотношение элементов конструкции,
структуры и структурного графа

Рис. 3.

Рис. 4. представляет структуру примерного мультипроцессора и его структурный граф.



Представление примерного мультипроцессора структурным графом

Рис. 4.

2.2 Описание тестовых процессов

Тест – это процедура, которая расценит элемент как нормальный (=0) или как ошибочный (=1). Отличительными характеристиками тестов предложенной модели являются

- подмножество структурных элементов, имеющих следующие типичные функции при выполнении теста:
  - произведение стимулей,
  - передача стимулей,

передача ответа,

анализ/оценка ответа и

организация и координация этих же функций;

- содействие, взнос теста к полному покрытию тестами тестированного элемента.

Таким образом тест t_i задается

- тест-графом $G(t_i)$, который является помеченным подграфом структурного графа G, при котором пометки с вышеуказанными функциями могут быть G_i, s_i, r_i, E_i, c_i по таблице 1;
- и присутствием теста в покрывающей функции некоторых структурных элементов.

Функции элементов структурного графа	Пометки тест-графа
произведение стимулей	G_i
передача стимулей	s_i
(тестированный элемент)	T_i
передача ответа	r_i
анализ/оценка ответа	E_i
организация/координация	c_i

Таблица I.

Необходимым условием детектируемости и диагностируемости любой системы является то, что из набора тестов можно было сгруппировать для каждого элемента структуры группу тестов полно покрывающую его.

2.3. Механизм инвалидации тестов

Результат теста может быть недействительным, неверным, если тестирующие элементы ошибочные – это называется инвалидацией. Были проведены мысленные эксперименты над примитивными тест-графами, из которых можно построить любые тест-графы, и на которые любые практические тест-графы можно разложить.

В результате установлено:

- вызывается инвалидация теста элемента, представленного вершиной, если любой из элементов, представленных вершиной с пометкой E или дугой между двумя вершинами с пометкой E, являются ошибочными;
- результат теста элемента, представленного вершиной, является функцией ИЛИ от состояния этого же элемента и элементов, представленных дугами с пометкой s и r;
- ошибки прочих, участвующих в тесте элементов инвалидацию не вызывают, но неопределенным на основе модели образом могут изменить с 0-я на 1-у результат теста элемента, представленного вершиной.

3. ДЕТЕКТИРУЕМОСТЬ

В предложенной модели мерой детектируемости (e) является то количество элементов структурного графа, соответствующих

ошибочным структурным элементам, при котором в случае любой комбинации ошибочных элементов ошибка системы обнаруживается, но есть такое множество e+1 элементов структурного графа, соответствующих ошибочным структурным элементам, при котором ошибка системы не обнаруживается.

Детектируемость зависит и от структуры S и от набора тестов $\mathcal{T}$ системы.

Были сформулированы и доказаны теоремы о детектируемости структуры S при некотором наборе тестов $\mathcal{T}$ – e (S,$\mathcal{T}$)–, о максимальной и мимнимальной возможной детектируемости структуры S при полных тестовых наборах – $e(S)_{max}$, $e(S)_{min}$.

Приведем последние две теоремы:

Теорема 1. Максимальная возможная детектируемость структуры S на одно меньше, чем количество вершин в наименьшей, сильно связанной такой компоненты графа структуры G, из которой не существует ориентированной цепи в другие компоненты:

$$e(S)_{max} = |N^{*}|_{min} - 1,$$

где N^{*} множество вершин вовне изолированного, сильно связанного подграфа.

Теорема 2. Максимальная возможная детектируемость структуры S полным на нее набором тестов на одно меньше, чем обхват – то есть длина кратчайшего ориентированного контура – в графе структуры G:

$$e(S)_{min} = g(G) - 1 .$$

Для систем с неполными наборами тестов $\mathcal{T}_o$ или для систем с нециклическим структурным графом ($g(G_o) = 0$) детектируемость:

$$e(S, \mathcal{T}_o) = e(S_o)_{max} = e(S_o)_{min} = 0.$$

4. ПРОЧИЕ РЕЗУЛЬТАТЫ И НАПРАВЛЕНИЕ ПРОДОЛЖЕНИЯ ИССЛЕДОВАНИЙ

Используя предложенную модель и разработанные теоремы легко решается задача по анализу и сравнению мультимикропроцессорных систем с точки зрения возможности самопроверки:

структурные варианты S_i и варианты наборов тестов $\mathcal{T}_j$ четко характеризуются показателями детектируемости

$$e(S_i)_{max},\ e(S_i)_{min},\ e(S_i, \mathcal{T}_j).$$

Теоремы и их доказательства показывают методы выявления"слабых и перестрахованных"точек структур и наборов тестов.

На основе модели была разработана также количественная мера наборов тестов.

Задача синтеза поставлена в работах [1] ÷ [4] как синтез структур и предложение оптимальных структур. Оптимальные структуры предлагаются и в настоящей модели, причем не только по параметрам $e(S)_{max}$, $e(S)_{min}$, но и по ожидаемому объему набора тестов и твердого ядра.

Задача синтеза автором поставляется скорее как синтез минимального набора тестов с предписанной детектируемости. (Тут речь идет о производстве спецификации набора тестов). Доказательства теорем показывают возможные методы, например, перечисление примитивных подграфов (тестов), покрывающих структурный граф.

Естественным продолжением этих работ является установление условий диагностируемости и разработка процедур анализа и синтеза по детектируемости и по диагностируемости.

Литература

[1] F. P. Preparata, G. Metze, R. T. Chien,
"On the connection assignment problem of diagnosable systems",
IEEE Trans. on Comp., Vol EC-16, No.6, Dec.1967., pp. 848-854.

[2] J. D. Russell, C. R. Kime,
"System fault diagnosis: closure and diagnosability with repair",
IEEE Trans. on Comp., Vol. C-24, No.11, Nov. 1975,
pp. 1078-1089.

[3] F. Barsi, F. Grandoni, P. Maestrini,
"A theory of diagnosability of digital systems",
IEEE Trans. on Comp., Vol. C-25, No.6. June 1976., pp.585-593.

[4] S. Karunanithi, A. D. Friedman,
"Analysis of digital systems using a new measure of system diagnosis",
IEEE Trans. on Comp., Vol. C-28, No.2, Feb. 1979., pp.121-133.

[5] Harmat L.,
"Diagnostic models for multiprocessor structures",
Preprints, Second Hungarian Comp. Science Conf.,
Budapest, 27 June - 2 July 1977., pp. 451-461.

[6] В. А. Гуляев,
"Организация систем диагностирования вычислительных машин",
Киев, Наукова Думка, 1979.

[7] P. Ciompi, L. Simoncini,
"Fault detection and diagnosis of digital systems: a review",
Proc. of Third Int. Seminar on Applied Aspects of Automata Theory, Varna, June 3-7, 1975.

[8] Harmat L.,
"Самодиагностизирующие структуры вычислительных машин",
часть I., (на венгерском языке),
Információ-Elektronika, Vol.XII., No.2, 1977., pp. 88-93.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

О декомпозиции коммутативных автоматов с помощью α_i-произведений.

Б. Имрех

В теории конечных автоматов большую роль играют разные способы композиции автоматов, позволяющие строить более сложные автоматы из более простых автоматов. Для таких рассмотрений были введены разные понятия композиций. Такие понятия: прямое произведение, квази-прямое произведение, α_i-произведение [7], произведение Глушкова [9] и обобщения этих понятий [8]. (С другой стороны известны разные представления автоматов как изоморфная реализация, гомоморфная реализация [9] и изоморфная симуляция, гомоморфная симуляция [8]. Для данной композиции и данного представления рассматривается следующий вопрос: как можно характеризовать системы конечных автоматов, из которых можно построить все конечные автоматы с помощью вибранных композиции и представления. Таким образом дается группа проблем, изображенная на следующей таблице. (Таблица заключает в себе ссылки на статьи, занимающиеся такими проблемами.)

Дальнейшие интересные вопросы даются при построении всех автоматов какого-нибудь специального класса конечных автоматов. Группа проблем, изобраменная в предыдущей таблице, интересна для класса конечных коммутативных автоматов. Исследования,

		реализация		симулясия	
		изоморфная	гомоморфная	изоморфная	гомоморфная
	прямое произведение				
	квази-прямое произв.	[5]			
	α_0-произведение		[1][2][3][4][14]		
	⋮				
	α_i-произведение	[10]			
	⋮				
	произв. Тушкова	[8]	[13]		
обобщенное	квази-прямое произв.				
обобщенное	α_0-произведение				
обобщенное	⋮				
обобщенное	α_i-произведение			[8]	[8]
обобщенное	⋮				
обобщенное	произв. Тушкова				

связанные с этим классом содержаются в рабоах [4], [6], [7], [14]. В дальнейших рассматривается вопрос, как можно характеризовать системы конечных автоматов, из которых можно постороить все конечные коммутативные автоматы с помощью α_i-произведения и изоморфной реализации. В этой работе мы дадим обзор о результатах работы [11], которые дадут ответ на этот вопрос.

Во первых мы вводим нужные понятия. <u>автоматом</u> называется объект $A=(X, A, \delta)$, где X -непустое конечное множество входных знаков, A -непустое конечное множество состояний и δ -отображение множества $A\times X$ в множество A. Пусть заданы автоматы $A=(X, A, \delta_A)$ и $B=(X, B, \delta_B)$. Автомат B называется <u>подавтоматом</u> автомата A если $B\subseteq A$ и функция δ_A совпадает с функцией δ_B на множестве $B\times X$. Взаимно однозначно отображение μ множества A на множество B называется <u>изо-</u>

<u>морфизмом</u> если выполняется $\mu(\delta_A(a,x)) = \delta_B(\mu(a),x)$ для любых элементов $a \in A$ и $x \in X$. Говорят, что автомат A <u>изоморфно реализует</u> автомат B если существует подавтомат A' автомата A такой, что автоматы A' и B являются изоморфными.

Обозначим через i фиксированное натуральное число и пусть заданы автоматы $A_t = (X_t, A_t, \delta_t)$ $(t=1,\dots,k)$. <u>α_i-произведение</u> автоматов A_t $(t=1,\dots,k)$ представляет собой автомат $A = (X, A, \delta)$, определенный заданием множества X и отображения φ множества $A_1 \times \dots \times A_k \times X$ в множество $X_1 \times \dots \times X_k$, где отображение φ выполняет следующие условия: $\varphi(a_1,\dots,a_k,x) = (\varphi_1(a_1,\dots,a_k,x),\dots,\varphi_k(a_1,\dots,a_k,x))$ и каждое отображение φ_j $(1 \le j \le k)$ зависит только от таких состояний, которые имеют индекс меньше чем $j+i$. Множество состояний A автомата A совпадает с произведением $A_1 \times \dots \times A_k$, а отображение δ определяется с помощью соотношения $\delta(a_1,\dots,a_k,x) = (\delta_1(a_1,\varphi_1(a_1,\dots,a_k,x)),\dots,\delta_k(a_k,\varphi_k(a_1,\dots,a_k,x)))$ для любых элементов $(a_1,\dots,a_k) \in A_1 \times \dots \times A_k$, $x \in X$.

Автомат $A = (X, A, \delta)$ называется <u>коммутативным</u> если выполняется соотношение $\delta(a,x_1,x_2) = \delta(a,x_2,x_1)$ для произвольных состояния $a \in A$ и входных знаков $x_1, x_2 \in X$. Обозначим через $\mathcal{K}$ класс всех коммутативных автоматов.

Автомат $A = (X, A, \delta)$ называется <u>связанным</u> если для произвольных состояний $a_1, a_2 \in A$ существуют входные слова p и q такие, что $\delta(a_1,p) = \delta(a_2,q)$. Обозначим через $\mathcal{K}_C$ класс всех коммутативных связанных автоматов.

Система автоматов Σ называется <u>изоморфно полной</u> для относительно α_i-произведения, если для любого автомата $A \in \mathcal{K}$

существует α_i-произведение автоматов из Σ, которое изоморфно реализует автомат A.

Во первых рассматривается α_0-произведение. Для коммутативных автоматов имеет место следующий результат.

Теорема. Любой коммутативный автомат можно изоморфно реализовать α_0-произведением автоматов из класса $\mathcal{K}_C$.

Для произвольного простого числа τ обозначим через $\bar{M}_\tau = (\{x_0,\dots,x_{\tau-1}\}, \{0,\dots,\tau-1\}, \delta_\tau)$ автомат, где $\delta_\tau(s,x_t) = s+t \pmod\tau$ и $\delta_\tau(v,x_\tau) = \delta_\tau(\tau,x_t) = \tau$ для любых s $(0\le s<\tau)$, t $(0\le t<\tau)$, v $(0\le v\le\tau)$. Пусть $\bar{M}$ -множество всех автоматов $\bar{M}_\tau$, где τ является простым числом. Пусть далее $E_2 = (\{x,y\},\{0,1\},\delta)$ автомат, где $\delta(0,y) = 0$ и $\delta(0,x) = \delta(1,x) = \delta(1,y) = 1$. Для класса $\mathcal{K}_C$ имеет место следующий результат.

Теорема. Любой коммутативный связанный автомат можно изоморфно реализовать α_0-произведением автоматов из класса $\bar{M}\cup\{E_2\}$.

Из определения α_0-произведения непосредственно вытекает, что α_0-произведение α_0-произведений является α_0-произведением. Таким образом из предыдущих теорем следует что система $\bar{M}\cup\{E_2\}$ изоморфно полна для $\mathcal{K}$ относительно α_0-произведения.

Применяя эти результаты не трудно доказать следующую теорему, содержащую необходимое и достаточное условие для того, чтобы система автоматов была изоморфно полна для $\mathcal{K}$ относительно α_0-произведения.

Теорема. Система автоматов Σ изоморфно полна для $\mathcal{K}$ относительно α_0-произведения тогда и только тогда, когда

выполняются следующие условия:

(1) Существует автомат $A_0 \in \Sigma$ такой, что автомат E_2 изоморфно реализуется α_0-произведением одного фактора автомата A_0.

(2) Для произвольного простого числа τ существует автомат $A \in \Sigma$ такой, что автомат $\overline{M_\tau}$ изоморфно реализуется α_0-произведением автоматов A_0 и A.

В дальнейших мы предположим, что $i \geq 1$ и рассмотрим α_i-произведение. Для произвольного простого числа τ обозначим через $M_\tau = (\{x_0, \ldots, x_{\tau-1}\}, \{0, \ldots, \tau-1\}, \delta_\tau)$ автомат, где $\delta_\tau(s, x_t) = s + t \pmod{\tau}$ для любых s $(0 \leq s \leq \tau-1)$, t $(0 \leq t \leq \tau-1)$. Пусть M - множество всех автоматов M_τ, где τ является простым числом. Тогда мы имеем следующую теорему.

Теорема Система автоматов Σ изоморфно полна для K относительно α_i-произведения тогда и только тогда, когда для любого простого числа τ существует автомат $A \in \Sigma$ такой, что автомат M_τ изоморфно реализуется α_i-произведением одного фактора автомата A.

Из этой теоремы вытекает, что системы, изоморфно полные для K относительно α_i-произведений совпадают для всех разных $i \geq 1$. С другой стороны, применяя результат работы [10] мы получим, что эти системы совпадают с системами, изоморфно полными для класса всех автоматов относительно α_i-произведений.

Литература

[1] Dilger, E., On Permutation-Reset Automata, Information and Control, 30 (1976), 86-95.

[2] Dömösi, P., On minimal R-complete systems of finite automata, Acta Cybernetica, 3 (1976), 37-41.

[3] Евтушенко, Н.В., К реализации автоматов каскадным соединением стандартных автоматов, АВТ, 2 (1979), 50-53.

[4] Fleck, A. C., Isomorphism groups of automata, J. Assoc. Comp. Machinary, 9 (1962), 496-476.

[5] Gécseg, F.,,On complete systems of automata, Acta Sci. Math., 30 (1969), 259-300.

[6] Gécseg, F., On subdirect representations of finite commutative unoids, Acta Sci.Math., 36 (1974),33-38.

[7] Gécseg, F., Composition of automata, Proceedings of the 2nd Colloquium on Automata, Languages and Programming, Saarbrücken, 1974, Springer Lecture Notes in Computer Science, V. 14, 351-363.

[8] Gécseg, F., On products of abstract automata, Acta Sci. Math., 38 (1976), 21-43.

[9] Глушков, В.М., Абстрактная теория автоматов, Успехи матем. наук, 16:5(101),1961,3-62.

[10] Imreh, B., On α_i-products of automata, Acta Cybernatica, 3 (1978), 301-307.

[11] Imreh, B., On isomorphic representations of commutative automata with respect to the α_i-products, Acta Cybernetica, to appear (1980).

[12] Krohn, K., and Rhodes, J., Algebraic theory of Machines. I Prime decomposition theorem for finite semigroups and machines, Trans. Am. Math. Soc. 116, 1965, 450-464.

[13] Летичевский, А. А., Условия полноты для конечных автоматов, Журнал вычисл. Матем. физ., 4 (1961), 702-710.

[14] Peák, I., Автоматы и полугруппа II, Acta Sci. Math. 26 (1965), 49-54.

[15] Zeiger, H.P., Cascade Synthesis of Finite-State Machines, Information and Control, 10 (1967), 419-433.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

ХАРТВИГ, Рольф ГДР

ЯЗЫКИ С ПЕРЕМЕННЫМИ С ИНДЕКСАМИ КАК ЧАСТИЧНЫЕ МНОГООСНОВНЫЕ ПЕАНО-АЛГЕБРЫ

R. Hartwig, Karl-Marx-Universität Leipzig, Sektion Mathematik, DDR-7010 Leipzig, Karl-Marx-Platz

В статье /2/ описана важная во многих отношениях возможность эквивалентного преобразования системы последовательно работающих присваиваний в оператор параллельных присваиваний ("параллельное предложение"). Существенным для вывода правил преобразований явилось описание действия параллельных присваиваний посредством соответствующих формальных подстановок. В качестве существенного достаточного условия для этого предположено наличие условных выражений в языке.

Излагаемый здесь алгебраический подход к синтаксису и к семантике произвольных языков с переменными с индексами, который следует некоторым мыслям Летичевского /I/, позволяет нетрудно доказывать также необходимость этого условия. Для этого общего подхода к синтаксису и к семантике языков с индексированными переменными надо расширить теорию частичных многоосновных алгебр, которая до сих пор мало развита.

Алгебраические основы

Тройку $\tau = (I, \Omega, \alpha)$ назовем многоосновной сигнатурой, если I и Ω – некоторые множества, а $\alpha : \Omega \longmapsto I^+$ – отображение множества Ω в множество I^+ всех конечных последовательностей элементов множества I.

$$\mathfrak{A} = [\ (A_i)_{i\in I},\ (f_\omega)_{\omega\in\Omega}\ ]$$

называется <u>частичной многоосновной алгеброй с сигнатурой</u> τ (короче, <u>частичной τ-алгеброй</u>), если $A = (A_i)_{i\in I}$ – семейство множеств, а $f = (f_\omega)_{\omega\in\Omega}$ – семейство частичных операций, где

$$f_\omega : A_{i_1} \times A_{i_2} \times \ldots \times A_{i_n} \rightarrowtail A_{i_{n+1}}$$

есть функция из $A_{i_1} \times \ldots \times A_{i_n}$ в $A_{i_{n+1}}$ в том случае, когда имеет место $\alpha(\omega) = (i_1,\ldots,i_n,i_{n+1})$. Элементы множества I называются <u>типами</u>, элементы множества Ω <u>операторами</u>, отображение α <u>I-арностью</u>, множества A_i называются <u>основными множествами</u> и функции f_ω – <u>операциями</u> алгебры $\mathfrak{A}$. Такую алгебру $\mathfrak{A}$ назовем <u>частичной многоосновной Пеано-алгеброй с сигнатурой</u> τ <u>и с многоосновным Пеано-базисом</u> X, если $X = (X_i)_{i\in I}$ и $X_i \subseteq A_i$ для любого $i\in I$ и если выполняются следующие обобщенные Пеано-аксиомы [1]:

П1. $f_\omega(a) \notin X_k$ для любого $\omega\in\Omega$ при $\alpha(\omega) = (i_1,\ldots,i_n,k)$ и для всех конечных последовательностей $a \in \mathbf{dom}\ f_\omega$ [2]

П2. $f_{\omega_1}(a) = f_{\omega_2}(b)$ заключает в себе $\omega_1 = \omega_2$ и $a = b$ для любых $\omega_1,\omega_2 \in \Omega$ и $a \in \mathbf{dom}\ f_{\omega_1}$, $b \in \mathbf{dom}\ f_{\omega_2}$.

П3. $[\ X\ ]^{\mathfrak{A}} = \mathfrak{A}$,

где $[X]^{\mathfrak{A}}$ обозначает подалгебру (наименьшее замкнутое подсемейство), порожденную подсемейством X.

Пусть $\mathfrak{B} = [\ (B_i)_{i\in I}, (g_\omega)_{\omega\in\Omega}\ ]$ – вторая частичная многоосновная алгебра с сигнатурой τ. Семейство $h = (h_i)_{i\in I}$ отображений $h_i : A_i \rightarrowtail B_i$ из A_i в B_i называется <u>конформным частичным τ-гомоморфизмом из</u> $\mathfrak{A}$ <u>в</u> $\mathfrak{B}$, если для всех

---

1) Сравни /3/ в однотипном случае.

2) $\mathbf{dom}\ f$ – область определения функции f.

$\omega\in\Omega$ при $\alpha(\omega) = (i_1,\ldots,i_n,i_{n+1})$ имеет место:

(1) $(a_1,\ldots,a_n) \in \mathrm{dom}\, f_\omega \wedge f_\omega(a_1,\ldots,a_n) \in \mathrm{dom}\, h_{i_{n+1}} \longleftrightarrow$

$a_1\in \mathrm{dom}\, h_{i_1} \wedge \ldots \wedge a_n \in \mathrm{dom}\, h_{i_n} \wedge (h_{i_1}a_1,\ldots,h_{i_n}a_n) \in \mathrm{dom}\, g_\omega$,

(2) $h_{i_{n+1}} f_\omega(a_1,\ldots,a_n) = g_\omega(h_{i_1}a_1,\ldots,h_{i_n}a_n)$.

Семейство $b = (b_i)_{i\in I}$ отображений $b_i : X_i \rightarrowtail B_i$ из базисных множеств X_i одной частичной многоосновной Пеано-алгебры $\mathfrak{A}$ в основные множества алгебры $\mathfrak{B}$ называется <u>частичным $\mathfrak{B}$-обложением</u> Пеано-базиса X алгебры $\mathfrak{A}$. Наконец, имеет место

<u>Теорема</u> (о продолжении). Для каждой частичной многоосновной Пеано-алгебры $\mathfrak{A}$ с сигнатурой τ, каждой частичной τ-алгебры $\mathfrak{B}$ и каждого частичного $\mathfrak{B}$-обложения b Пеано-базиса X алгебры $\mathfrak{A}$ существует один и только один конформный частичный τ-гомоморфизм h_b из $\mathfrak{A}$ в $\mathfrak{B}$, который над X тождественно совпадает с b, включая "неопределенного значения".

Теорему можно доказывать рекурсивно посредством принципа алгебраической индукции, который переносится на частичные многоосновные алгебры.

<u>Языки с переменными с индексами</u>

Синтаксис языков термов с переменными с индексами (короче: "пи-языков") можно описывать удобно посредством частичных двухтипных Пеано-алгебр со специальной сигнатурой. Сигнатура $\Delta = (I_\Delta, \Omega_\Delta, \alpha_\Delta)$ называется <u>япи-сигнатурой</u>, если I_Δ - двухэлементное множество (пусть здесь просто $I_\Delta = \{1, 2\}$) и для каждого $\omega\in\Omega_\Delta$ выполняется: из $\alpha_\Delta(\omega) = (i_1,\ldots,i_n,i_{n+1})$ следует, что $n = 0$ или $i_1=i_2= \ldots =i_n=1$. В зависимости

от того, что $i_{n+1} = 1$ или $i_{n+1} = 2$, пишем $\omega \in \Phi$ или $\omega \in \Lambda$, так что для япи-сигнатур Δ имеет место:

$$\Omega_\Delta = \Phi \cup \Lambda \quad , \text{ где } \Phi \cap \Lambda = \emptyset .$$

Поэтому задаем частичную Δ-алгебру $\mathfrak{A} = [(A_i)_{i \in I_\Delta}, (f_\omega)_{\omega \in \Omega_\Delta}]$ с япи-сигнатурой Δ четверкой

$$\mathfrak{A} = [\, A_1 , A_2 ; (f_\varphi)_{\varphi \in \Phi} , (f_\lambda)_{\lambda \in \Lambda} \,] .$$

Множество **EXPR** назовем <u>языком (термов) с переменными с индексами</u>, если **EXPR** есть основное множество первого типа частичной многоосновной Пеано-алгебры

$$\mathbf{SYN} = [\, \mathbf{EXPR} , \mathbf{VAR} ; \mathfrak{H} , \mathfrak{V} \,]$$

с япи-сигнатурой Δ , для Пеано-базиса $X = (X_1, X_2)$ которой выполняется $X_1 = \mathbf{VAR}$. При этом алгебра **SYN** называется <u>языко-определяющей</u> или <u>синтаксической алгеброй</u>, элементы множества **EXPR** называются <u>выражениями</u> языка, а элементы множества **VAR** <u>переменными</u>. Семейства $\mathfrak{H} = (f_\varphi)_{\varphi \in \Phi}$ и $\mathfrak{V} = (f_\lambda)_{\lambda \in \Lambda}$ называются семействами <u>словарных функций, порождающие выражения</u> и соответственно <u>переменные</u>. X_2 обозначается множеством **SIMPLV** <u>простых переменных</u>, а множество **CEXPR** = $= \bigcup_{\varphi \in \Phi} \mathbf{im}\, f_\varphi$ [1] называется множеством <u>составных выражений</u> и множество **SUBSV** = $\bigcup_{\lambda \in \Lambda} \mathbf{im}\, f_\lambda$ множеством <u>переменных с индексами</u>.

Из свойств частичных многоосновных Пеано-алгебр следует

<u>Теорема</u> (о пи-языках) .

(1) $\mathbf{SIMPLV} \subseteq \mathbf{VAR} \subseteq \mathbf{EXPR}$

(2) $\mathbf{SIMPLV} \cap \mathbf{SUBSV} = \emptyset \qquad \mathbf{VAR} = \mathbf{SIMPLV} \cup \mathbf{SUBSV}$

---

[1] **im f** – область значений функции

(3) $\mathbf{VAR} \cap \mathbf{CEXPR} = \emptyset$ $\qquad\qquad$ $\mathbf{EXPR} = \mathbf{VAR} \cup \mathbf{CEXPR}$

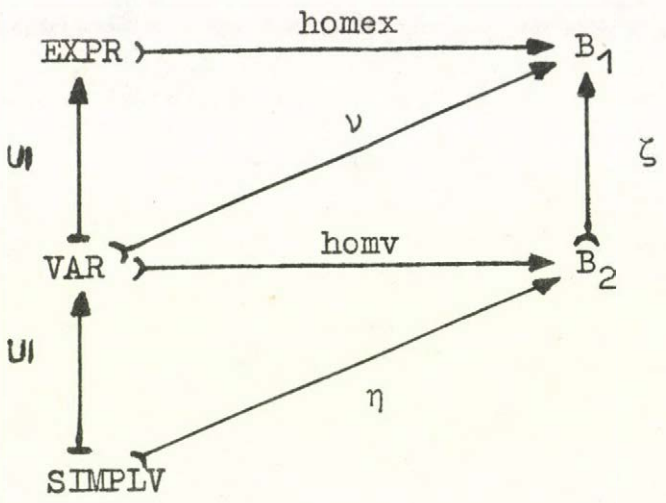
(4) $\forall \omega_1 \omega_2 \mathcal{H} \mathcal{H}' (\omega_1, \omega_2 \in \Phi \cup \Lambda \wedge \mathcal{H} \in \mathrm{dom}\, \mathfrak{f}_{\omega_1} \wedge \mathcal{H}' \in \mathrm{dom}\, \mathfrak{f}_{\omega_2}$
$\wedge\ \mathfrak{f}_{\omega_1}(\mathcal{H}) = \mathfrak{f}_{\omega_2}(\mathcal{H}') \longrightarrow \omega_1 = \omega_2 \wedge \mathcal{H} = \mathcal{H}')$

В дальнейшем фиксируем япи-сигнатуру Δ и рассматриваем конформные Δ-гомоморфизмы в классе всех частичных Δ-алгебр. Если **EXPR** есть язык с переменными с индексами и **SYN** = $= [\ \mathbf{EXPR}\ ,\ \mathbf{VAR}\ ;\ \mathfrak{H}\ ,\ \mathfrak{B}\ ]$ - его определяющая алгебра, то по теореме о продолжении для каждой частичной Δ-алгебры $\mathfrak{B} = [\ B_1\ ,\ B_2\ ;\ (g_\varphi)_{\varphi \in \Phi}\ ,\ (g_\lambda)_{\lambda \in \Lambda}\ ]$ и для каждого частичного $\mathfrak{B}$-обложения (ν, η) Пеано-базиса **(VAR , SIMPLV)** алгебры **SYN** существует однозначно конформный частичный Δ-гомоморфизм **hom = (homex, homv)** из **SYN** в $\mathfrak{B}$, который является естественным продолжением пары отображения (ν, η) . В этом контексте отображение ν назовем <u>значением переменных</u>, а отображение η <u>наименованием простых переменных</u>.

Алгебра **SYN** особенно характеризована тем, что между ее основными множествами существует отношение, а именно $\mathbf{VAR} \subseteq \subseteq \mathbf{EXPR}$. Поэтому нас интересуют особенно такие образы $\mathfrak{B}$ алгебры **SYN**, для основных множеств которых существует тоже некоторое соответствие. Назовем пару $[\mathfrak{B}\ ,\ \zeta]$ <u>связанной Δ-алгеброй</u>, если $\mathfrak{B}$, как раньше, есть частичная Δ-алгебра, и

$$\zeta : B_2 \rightarrowtail B_1$$

- отображение из B_2 в B_1. Назовем ζ <u>связующим отображением</u> в $\mathfrak{B}$. Принимать во внимание заданное связующее отображение, это значит, рассматривать только такие $\mathfrak{B}$-обложения (ν, η) , у которых следующая диаграмма коммутативна.



Существование таких $\mathfrak{B}$-обложений для любых связанных Δ-алгебр $[\mathfrak{B}, \zeta]$ обеспечивается следующей теоремой.

Теорема (об образах пи-языков). Для каждой синтаксической алгебры SYN пи-языка, каждой связанной Δ-алгебры $[\mathfrak{B}, \zeta]$ и каждого наименования простых переменных η : SIMPLV $\rightarrowtail B_2$ существует однозначно определенное значение переменных ν : VAR $\rightarrowtail B_1$, так что естественное продолжение hom = = (homex, homv) $\mathfrak{B}$-обложения (ν, η) выполняет равенство

$$\nu = \zeta \circ \text{homv} .$$ //

Из этого следует, что конформный частичный Δ-гомоморфизм hom = (homex, homv) однозначно определен уже через η и ζ.

Вообще говоря, можно принимать каждую связанную Δ-алгебру $[\mathfrak{B}, \zeta]$ за семантику рассматриваемого пи-языка, или наоборот, семантика пи-языков можно задавать простым способом в виде связанной Δ-алгебры [SEM, r] . Таким образом

$$\text{SEM} = [\, W , N ; \mathfrak{F} , \mathfrak{G} \,]$$

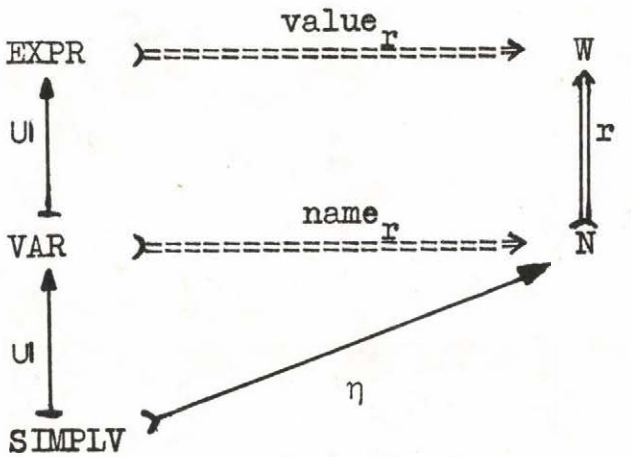
называется семантической алгеброй, W - множеством значений,

$\mathbf{N}$ – множеством имен, $\mathfrak{F} = (\psi_\varphi : W^{n_\varphi} \rightarrowtail W)_{\varphi \in \Phi}$ – семейством семантических операций, и $\mathfrak{G} = (\sigma_\lambda : W^{n_\lambda} \rightarrowtail N)_{\lambda \in \Lambda}$ – семейством выбирающих функций. Частичное отображение

$$r : N \rightarrowtail W$$

назовем референтивным отображением или отношением "именовать". В зависимости от наименования η, которое в дальнейшем считаем фиксированным, и от референтивного отображения r получаем семантический гомоморфизм $sem_r = (value_r, name_r)$ как естественное продолжение hom.

Получается следующая коммутативная диаграмма



и выполняются следующие рекурсивные равенства

$value_r(v) = r(\eta(v)),\ name_r(v) = \eta(v)$ для $v \in SIMPLV$,

$value_r(f_\lambda(H)) = r(name_r(f_\lambda(H))),\ name_r(f_\lambda(H)) = \sigma_\lambda(value_r(H))$ для $\lambda \in \Lambda$ и $H \in dom\ f_\lambda$,

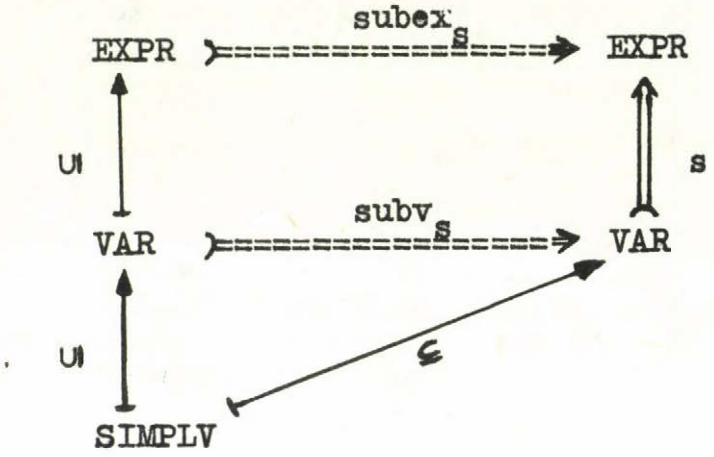
$value_r(f_\varphi(H)) = \psi_\varphi(value_r(H))$ для $\varphi \in \Phi$ и $H \in dom\ f_\varphi$.

Если выбираем $SEM = SYN$ как семантическую алгебру и тождественное отображение ι, где $\iota(v) = v$ для всех $v \in SIMPLV$, как наименование простых переменных η, то получаем важный специальный случай. Таким образом, каждое связующее отобра-

жение в **SYN** представляет собой подстановку

$$s : VAR \rightarrowtail EXPR$$

в пи-языке **EXPR** , и однозначно определенное через **s** (и ι) продолжение **hom = (homex,homv)** описывает выполнение этой подстановки. Оно называется поэтому индуцированным через **s** эндоморфизмом подстановки $sub_s = (subex_s, subv_s)$. Получаем следующую коммутативную диаграмму



и следующие рекурсивные равенства

$subex_s(v) = s(v)$, $subv_s(v) = v$ для $v \in SIMPLV$,

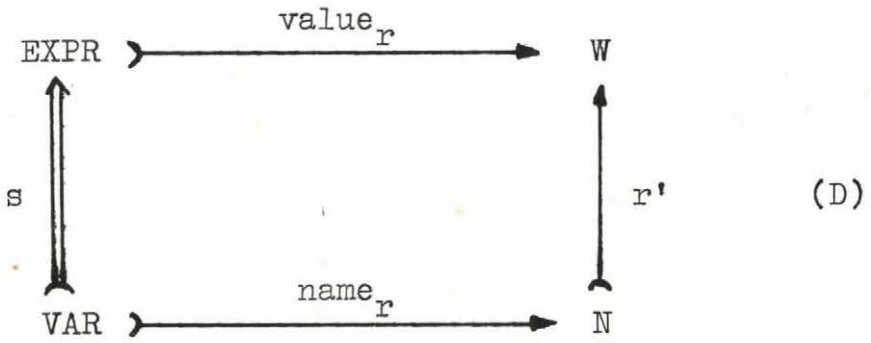
$subex_s(f_\lambda(H)) = s(subv_s(f_\lambda(H)))$, $subv_s(f_\lambda(H)) = f_\lambda(subex_s(H))$ для $\lambda \in \Lambda$ и $H \in dom\ f_\lambda$,

$subex_s(f_\varphi(H)) = f_\varphi(subex_s(H))$ для $\varphi \in \Phi$ и $H \in dom\ f_\varphi$.

Параллельные присваивания и подстановки

Параллельное присваивание является синтаксическим указанием некоторого отношения "именовать". Согласно заданному предписанию (сравни например /2/, /4/) сопоставляются значения, представляемые выражениями, именам, представляемые переменными.

Возникают вопрос, существует ли при любой семантике [SEM, r] для каждого параллельного присваивания подстановка s , которая действует таким же образом. Если r' обозначает описанное параллельным присваиванием реферативное отображение, то можно алгебраически формулировать вышеуказанный вопрос следующим образом: Существует ли подстановка s , для которой следующая диаграмма коммутативна:



При достаточно сложной семантической алгебре $\mathfrak{A}$ вытекает существование "условных выражений" в языке EXPR в том случае, когда такая подстановка s существует для всех r и r' . При этом назовем выражение H^* условным выражением (со сравнением имен), если существуют переменные $u,v \in VAR$ и выражения $H_1, H_2 \in EXPR$, для которых при любых реферативных отображениях r выполняется

$$value_r(H^*) = \begin{cases} value_r(H_1) , \text{ если } name_r(u) = name_r(v) \\ value_r(H_2) , \text{ если } name_r(u) \neq name_r(v) \end{cases},$$

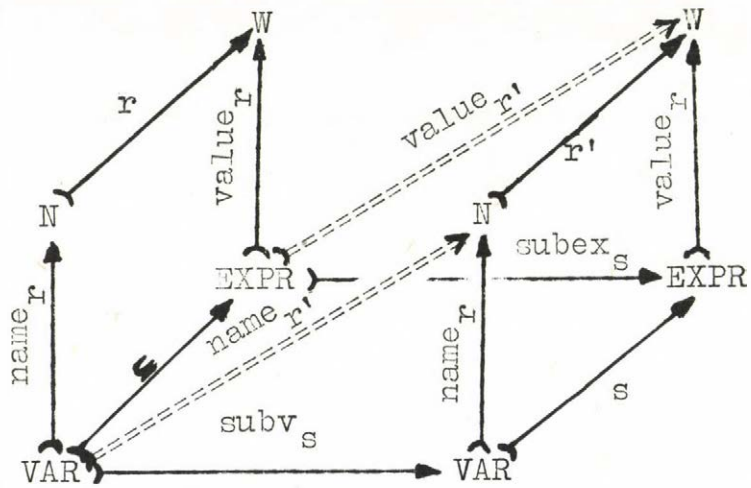
но H^*, H_1 и H_2 – попарно не равны для всех значений. Вышеуказанная диаграмма (D) – коммутативна только тогда, когда для всех переменных $u,v \in VAR \cap dom\ s$ из равенства $name_r(u) = name_r(v)$ всегда следует равенство $value_r(s(u)) = value_r(s(v))$. Это выполняется в общем

случае только тогда, когда выражения s(u) и s(v) являются условными.

Этим имеем одно направление доказательства следующей теоремы.

Теорема. В нетривиальных языках с переменными с индексами существование условных выражений является необходимым и достаточным для того, чтобы синтаксически отражать произвольный параллельным присваиванием описанный переход реферативного отображения в другое подстановкой.

Наоборот, с диаграммой (D) имеем тоже следующую коммутативную диаграмму



из которой можем читать равенства, названы в /2/ условиями корректности,

$$value_{r'}(H) = value_r(subex_s(H)) \qquad \text{для } H \in EXPR ,$$

и

$$name_{r'}(v) = name_r(subv_s(v)) \qquad \text{для } v \in VAR .$$

С этими равенствами также показана достаточность наличия условных выражений для утверждения теоремы.

Л И Т Е Р А Т У Р А

/I/ Летичевский, А. А., Синтаксис и семантика формальных языков, Кибернетика 4/68, I - 9

/2/ Хартвиг, Р., О преобразовании последовательной строки операторов присваивания в параллельную, с учетом индексации, В сб.: MTA SZTAKI Tanulmányok, Budapest 1981

/3/ Burmeister,P., Schmidt, J., On the completion of partial algebras, Coll. Math. 17(1967), 235 - 245

/4/ Hartwig, R,, Über sprachliche Ausdrucksmittel zur syntaktischen Beherrschung der Äquivalenz von Folgen simultaner Wertzuweisungen an indizierte Variablen, Elektronische Informationsverarbeitung und Kybernetik (в печати)

/5/ Hartwig, R., Syntax und Semantik von Sprachen mit indizierten Variablen, Preprint Sekt. Math. KMU Leipzig 1980

MTA Számítástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

ПРАМАЯ ОБРАБОТКА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ ОДНОРОДНЫМИ СТРУКТУРАМИ

Й. Метц

Технический Университет Дрезден, ГДР

Сообщение посвящено проектированию программируемых схем, прямо имплементирующих языки программирования для задач управления. Выбранная основная структура дает возможность высокой параллельности и скорости обрабатывающих процессов. В самом простом случае предполагается регулярная грамматика языков, причем семантика описана автоматными функциями. Обсуждаются пример и принадлежащий вариант реализации.

На рис.I представлена основная структура. Она является двумерно бесконечной сетью автоматов, построенной из автоматов A_0, A_I, A_2 и A_3. Входы a_1, a_2, ... принимают знаки программы. Число необходимых в конкретном случае в интерпретирующей схеме автоматов A_I определено длиной программы. Внутренние переменные z_1, z_2, ... могут получить значения входов x_1, x_2, ... Выходами y_1, y_2, ... выдаются актуальние значения переменных z_1, z_2, ... , которые сохраняются в автоматах A_2. Число автоматов A_2 в конкретной реализации соответствует число этих переменных.

Пусть $G = (A, H, s, R)$ - грамматика языка программирования, причем A терминальный алфавит, H спомогательный алфавит, $s \in S$ начальный символ и $R \subseteq H \times (A\,H \cup \{e\})$ множество правил. Символом e обозначим пустое слово. Комбинаторные автоматы A_I симулируют вывод программы в соответствии с грамматикой языка. Требуется, что из $(h, ah'), (h, ah'') \in R$ следует $h' = h''$ для всех $h, h', h'' \in H$ и $a \in A$. Так существует частичная функция $h_{j+1} := \alpha(h_j, a_j)$ автоматов A_I, причем $(h_j, a_j h_{j+1}) \in R$ для $h_j \in H$, $a_j \in A$, $j=1,2,\ldots$.

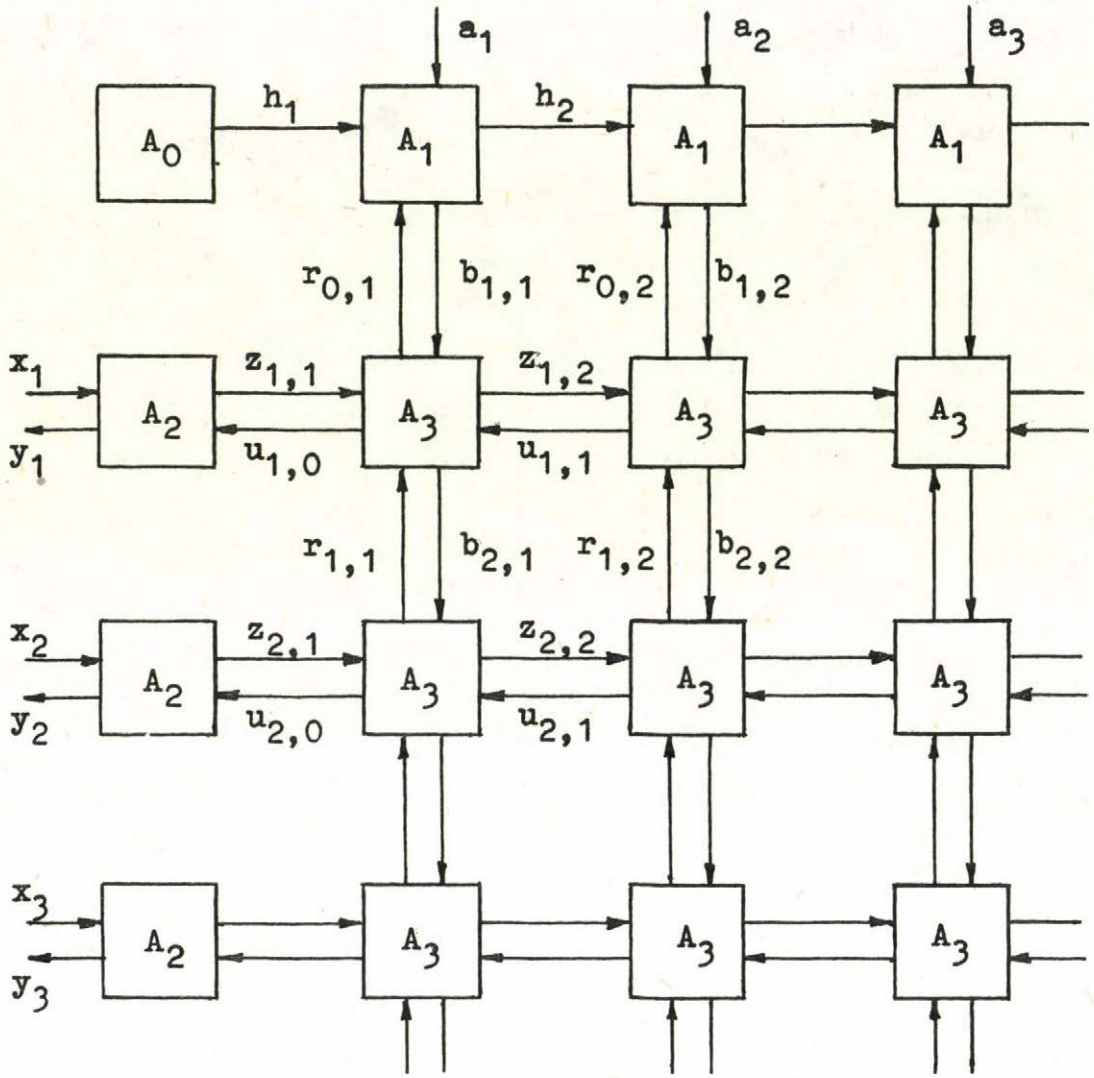

a1
a2
a3
A0
h1
A1
h2
A1
A1
r0,1
b1,1
r0,2
b1,2
x1
A2
z1,1
A3
z1,2
A3
A3
y1
u1,0
u1,1
r1,1
b2,1
r1,2
b2,2
x2
A2
z2,1
A3
z2,2
A3
A3
y2
u2,0
u2,1
x3
A2
A3
A3
A3
y3

Рис. I

Единственной задачей автомата A_0 является предоставление начальный символ $h_1 = s$. Программа $a_1 a_2 \ldots a_m$ называется допустимой, если $(h_{m+1}, e) \in R$.
Кроме того, автоматы типа A_I вызывают интерпретацию программ комбинаторными автоматами A_3. Для этого, прежде всего, каждый автомат A_3 передает вверх информацию

$$r_{i-1,j} := \beta(r_{i,j}, z_{i,j})$$

$(i,j = 1,2,\ldots)$. Чтобы обеспечивать вычисление всех $r_{i,j}$ условливается, что $z_{i,j} = \varepsilon$ если $i > n$ и n называет число переменных, используемых в конкретной программе. Потом требуется $\beta(r_{i,j}, \varepsilon) = \varepsilon$. Автоматы A_I реализуют инструкции $b_{1,j} := \gamma(h_j, a_j, r_{0,j})$, которые перерабатываются автоматами A_3 следующим образом:

$$z'_{i,j} := \delta(r_{i,j}, z_{i,j})$$
$$b_{i+1,j} := \lambda(b_{i,j}, z'_{i,j})$$
$$z_{i,j+1} := \mu(b_{i,j}, z'_{i,j})$$

Далее требуются $\gamma(h, \varepsilon, r) = \delta(r, \varepsilon) = \lambda(b, \varepsilon) = \lambda(\varepsilon, z') = \mu(b, \varepsilon) = \mu(\varepsilon, z') = \varepsilon$ для всех h, r, b и z', чтобы создать вне активной части поля единые и ясные условие. С помощью следующих словарных функций можно описать действие принадлежащих к правилам грамматики инструкций:

$$\underline{\beta}(r,e) = r\,, \quad \underline{\beta}(r,\underline{z}z) = \underline{\beta}(\beta(r,z),\underline{z})$$
$$\underline{\delta}(r,e) = e\,, \quad \underline{\delta}(r,\underline{z}z) = \underline{\delta}(\beta(r,z),\underline{z})\,\delta(r,z)$$
$$\underline{\mu}(b,e) = e\,, \quad \underline{\mu}(b,\underline{z}z) = \mu(b,z)\,\underline{\mu}(\lambda(b,z),\underline{z})$$

Символом $\underline{z}$ обозначим слово состоящее из знаков z_1, z_2, ... Получена интерпретация правила $(h, ah') \in R$:

$$z_1 z_2 \ldots z_m \to \underline{\mu}(\gamma(h,a,\underline{\beta}(\varepsilon, z_1 z_2 \ldots z_m)), \underline{\delta}(\varepsilon, z_1 z_2 \ldots z_m))$$

Таким образом программа знак за знаком обрабатывается. Результат $z_{1,m}$, $z_{2,m}$, ..., $z_{n,m}$ передается обратно к автоматами A_2 при помощи специального набора проводов и через автоматы A_3:

$$u_{i,j-1} := \begin{cases} u_{i,j} & \text{если } b_{i,j} \neq \varepsilon \\ z_{i,j} & \text{если } b_{i,j} = \varepsilon \end{cases}$$

Если сеть автоматов работает синхронно по тактам, то программа может быть обработана повторно. Только автоматы A_2

имеют память и могут получить или выдавать внешние информации.

Примером является следующий язык программирования для одновременного вычисления рядов инструкции с Булевыми дизъюнктивными нормальными формами.

<программа> ::= begin <ряд инструкций> end
<ряд инструкций> ::= <инструкция> | <ряд инструкций>
; <инструкция>
<инструкция> ::= <переменная> := <выражение>
<выражение> ::= <конюнкция> | <выражение> ∨
<конюнкуия>
<конюнкуия> ::= <переменная> | <конюнкция> ∧
<переменная>
<переменная> ::= <цифра> | ¬ <цифра> |
<переменная> <цифра>
<цифра> ::= 0 | I | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Переменными являются ряды цифр. Если переменная с отрицанием на левой стороне инструкции, то она получает отрицательное значение выражения правой стороны.
Функция автомата A_I определена на таблице I. Есть и две дополнительные бинарные величины v_j и w_j. Они соответствуют дополнительной строке, которая реализована в автоматы A_I. Для того, чтобы обработка упростится программа дана в обратном порядке. Табл.2 определены автоматы A_3. Величины $z_{i,j}$ разлагаются в две бинарные компоненты $z_{1,i,j}$ и $z_{2,i,j}$, причем $z_{2,i,j}$ является маркой. Эта таблица может быть использована для оценки затраты. Оказывается, что современной технологией около 10^2 до 10^3 автоматов A_3 могут реализироваться в одном элементе с высоко интегрируемой логикой. Для практических применений этот предел нам кажется еще мало для того, чтобы целое поле укладываться в один элемент. У каждого автомата A_3 16 входов и выходов. Если совместно будут реализованы 20 автоматов типа A_3, то получм элементы с 70 входами и выходами, с помощью которых может быть реализовано поле любого размера.

h_j	a_j	w_j	h_{j+1}	$b_{1,j}$	v_{j+1}	w_{j+1}
1	<u>end</u>		2	11	1	0
2	0 1 ⋮ 9		3	0 1 ⋮ 9	v_j	w_j
3	0 1 ⋮ 9		3	0 1 ⋮ 9	v_j	w_j
	¬		4	10	v_j	w_j
	∧		2	11	$v_j \wedge r_{0,j}$	w_j
	∨		2	11	1	$w_j \vee (v_j \wedge r_{0,j})$
	:=		5	11	1	$w_j \vee (v_j \wedge r_{0,j})$
4	∧		2	11	$v_j \wedge \neg r_{0,j}$	w_j
	∨		2	11	1	$w_j \vee (v_j \wedge \neg r_{0,j})$
	:=		5	11	1	$w_j \vee (v_j \wedge \neg r_{0,j})$
5	0 1 ⋮ 9		6	0 1 ⋮ 9	v_j	w_j
6	0 1 ⋮ 9		6	0 1 ⋮ 9	v_j	w_j
	¬		7	10	v_j	w_j
	;	0 1	2	12 13	1	0
	<u>begin</u>	0 1	8	12 13	1	0
7	;	0 1	2	13 12	1	0
	<u>begin</u>	0 1	8	13 12	1	0
8						

Табл. I

$z_{2,i,j}$	$b_{i,j}$	$r_{i-1,j}$	$b_{i+1,j}$	$z_{1,i,j+1}$	$z_{2,i,j+1}$	$u_{i,j-1}$
0	0	$r_{i,j}$	0	$z_{1,i,j}$	0	$u_{i,j}$
	1		1		0	
	2		2		0	
	3		3		0	
	4		4		0	
	5		5		0	
	6		6		0	
	7		7		0	
	8		8		0	
	9		9		0	
	10		10		0	
	11		11		1	
	12		12		1	
	13		13		1	
1	0	$z_{1,i,j}$	9	$z_{1,i,j}$	1	$u_{i,j}$
	1		0		0	
	2		1		0	
	3		2		0	
	4		3		0	
	5		4		0	
	6		5		0	
	7		6		0	
	8		7		0	
	9		8		0	
	10		10		1	
	11		11		1	
	12		11		1	0
	13		11		1	1

Табл 2

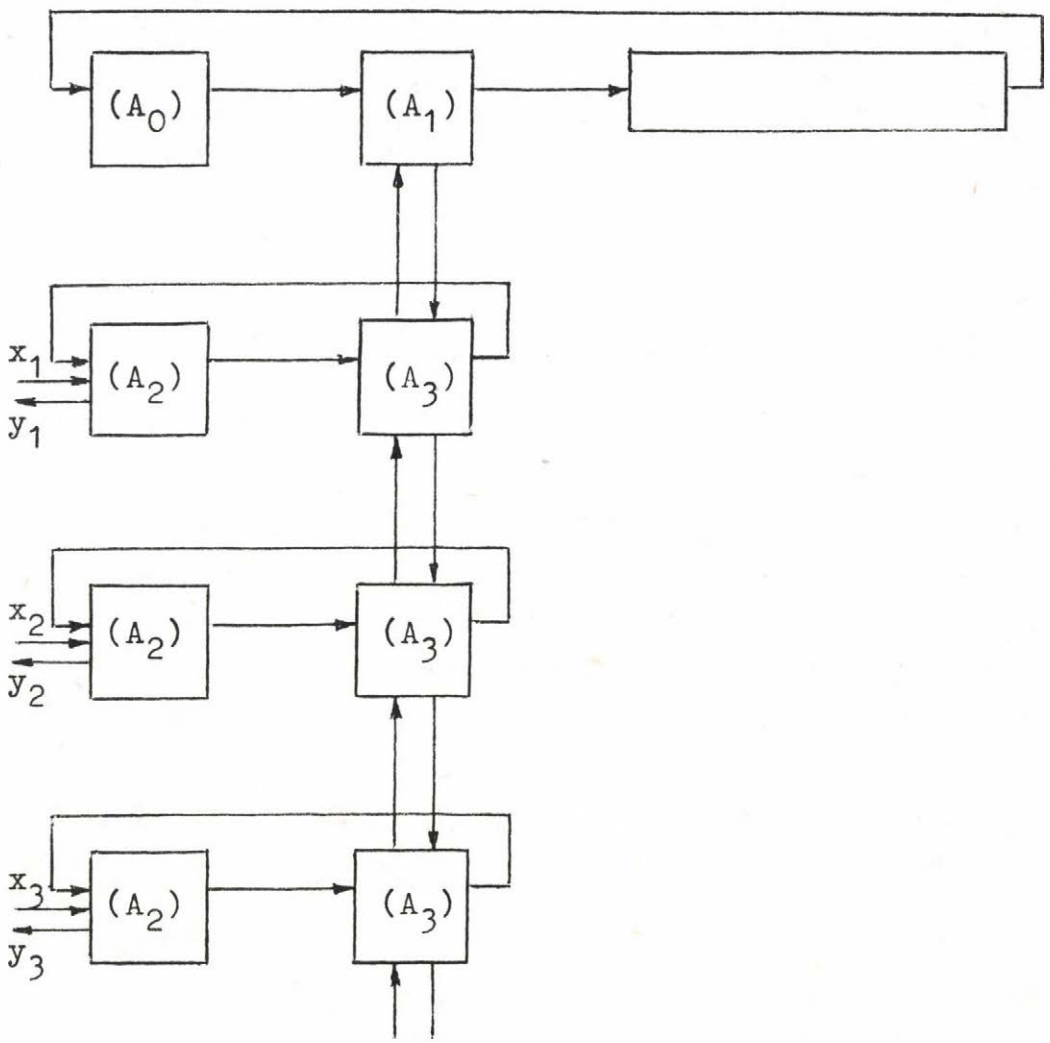


Рис. 2

Более удобним вариянтом является реализация по рис.2, которая получена по тому же самому принципу проектировании как прежде, но базируется на последовательной и цикличной обработки программы с помощью динамичечкой памяти. В особенности тогда, когда не надо вывести все переменных, могут быть указываны структуры, которыми обеспечиваются большая приспособляемость и хорошое использование достижимой степени интегрирования.

Представленный принцип проектирования можно и применить при реализации других примеров языков программирования. Ограничение на языкы с регулярной грамматикой не является существенным. Любой язык, акцептированный детерминированным многоленточном автоматом в реальное время, может быть интерпретирован с помощью моделирования лентов на поле. Однако с ростом сложности сети более значительными становятся проблемы задержки. В представтенном примере граница задержки по порядку равна сумме $n + m$ /число переменных и знаков в программе/, и не растет так сильно как произведение $n \cdot m$.

Лит.: Metz, J., Lütjens, G., Homogene unendliche Automatennetze aus Mealy-Automaten, IFAC-Symposium Diskrete Systeme, Dresden 1977, 5, 83 - 93.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

ON THE REPRESENTATION OF FINITE LATTICES IN THE CLASS OF FINITE AUTOMATA

Jerzy W. Grzymala-Busse

Introduction

An important part of the automata theory is the structure theory of finite automata, which deals mainly with the problems of some types of decomposition of finite automata /9/. The basic tool of the structure theory of automata is the concept of a partition on the set of states of the finite automaton with the substitution property, or shortly a S.P. partition.

In the paper we concern such finite automata that their nontrivial S.P. partitions are incomparable. In other words, we consider such automata that their product and sum of any two nontrivial S.P. partitions are equal to 0 and 1, respectively, where 0 is the trivial partition such that each state constitute one-element block of it and 1 is the trivial partition such that it contains just one block. It is an important case for applications, since for a finite automaton A there exists a realization in the form of parallel connection of two simpler automata if and only if there exist two S.P. partitions such that their product is equal to 0 /9/. We study a modified problem from /5/: Is it possible to represent arbitrary finite lattice with incomparable nontrivial members in the form of the lattice of all S.P. partitions of some finite automaton? This problem is still open. There exists more general open prolem /11/, known as extremely difficult: Is it possible to represent a finite lattice in the form of the congruence lattice of a finite abstract algebra? These problems are similar to another one solved in /7/: For given monoid E of functions to find a set of all automata with the endomorphism monoid equal to E.

In the paper we study in detail the class of total automata and as a result we have that in this class there exist automata with p+1 incomparable nontrivial S.P. partitions only, where p is a prime number. Moreover, it is shown that in the class of incompletely specified automata can be represented arbitrary finite lattice with incomparable nontrivial members.

1. Preliminary Definitions and Results

We quote a few definitions, which can be found e.g. in /3,4,9/.

A finite automaton A (or briefly automaton) is a pair (S,J), where S is a finite nonempty set (called also a state set) and J is a nonempty set of functions of S into S. The set J, together with the operation of superposition, generates a semigroup $\langle J \rangle$ of functions of S into S. Any element of J will be written as a right operator. For $f, f' \in \langle J \rangle$ we have

$$s(ff) = (sf)f'$$

An automaton A = (S,J) is said to be connected if and only if for any $s, s' \in S$, there exist a sequence $s = s_0, s_1, \ldots, s_n = s'$ of elements of S, and a sequence $f_0, f_1, \ldots f_{n-1}$ of elements of $\langle J \rangle$ such that either $s_i f_i = s_{i+1}$ or $s_{i+1} f_i = s_i$ for $i = 0,1,\ldots,n-1$. An automaton A = (S,J) is said to be strongly connected if and only if for any $s,s' \in S$ there exists $f \in \langle J \rangle$ such that $sf = s'$.

A partition π on S is a collection of mutually disjoint subsets of S whose union is S. These disjoint subsets of S will be called blocks of π. For $s,s' \in S$ we shall write $s \equiv s'(\pi)$ if and only if both s and s' are members of the same block of π.

Let π and π' be partitions on S. Then the product of partitions π and π' is the partition $\pi \cdot \pi'$ such that $s \equiv s'(\pi \cdot \pi')$ if and only if $s \equiv s'(\pi)$ and $s \equiv s'(\pi')$. The sum of partitions π and π' is the partition $\pi + \pi'$ such that $s \equiv s'(\pi + \pi')$ if and only if there exists a sequence $s = s_0, s_1, \ldots, s_n = s'$ of elements of S such that $s_i \equiv s_{i+1}(\pi)$ or $s_i \equiv s_{i+1}(\pi')$ for $i = 0,1,\ldots,n-1$. For partitions π and π' on S we say that π is larger than or equal to π', and write $\pi' \leq \pi$, if and only if each block of π' is contained in a block of π. We say also that π and π' are comparable. If for π and π' we have that neither $\pi \leq \pi'$ nor $\pi' \leq \pi$ then we say that π and π' are incomparable.

For the automaton A = (S,J), a partition π on S has the substitution property if and only if $s \equiv s'(\pi)$ for $s,s' \in S$ implies that $sf \equiv s'f(\pi)$ for any $f \in J$. We refer briefly to partitions with the substitution property on A as S.P. partitions on A. The set of all S.P. partitions on A,

together with the operations + and · , forms a lattice L(A).

PROPOSITION 1.1. For any automaton A = (S,J) there exists connected automaton B = (S,K) such that $J \subseteq K$ and L(A) = L(B).

PROOF. Let s be a fixed member of S and let f be a function of S into S defined for any $t \in S$ as follows f(t) = s. Let $K = J \cup \{f\}$ and let B = (S,K). Let π be arbitrary member of L(A). First, from the definition of the substitution property on A it follows that $\pi \in L(B)$ if $r \equiv r'(\pi)$ implies $rf \equiv f'f(\pi)$ for any $r,r' \in S$. Moreover, we have that rf = s = rf', and hence the last implication is always satisfied. Finally, the fact that the automaton B is connected follows directly from the definition of connectedness.

PROPOSITION 1.2. For any automaton A = (S,J) there exists strongly connected automaton B = (S,K) such that $J \subseteq K$ and L(A) = L(B).

PROOF. For some $s \in S$ let f_s be a function such that for any $t \in S$ we have $f_s(t) = s$. Let R be a set of functions f_s for all $s \in S$. Let B be an automaton (S,K), where $K = J \cup R$. The rest of the proof is just a verification of the fact that B is strongly connected and L(A) = L(B).

From Proposition 1.2 it follows that in finding a representation of the finite lattice as the lattice of all S.P. partitions on some automaton we can restrict ourself - without loss of generality - to the class of all strongly connected automata.

For the rest of the paper we shall consider the problem of finding a class G_n of all automata A such that L(A) has exactly n incomparable nontrivial S.P. partitions on A, where n is arbitrary non-negative integer. G_0 denotes the class of all automata with trivial S.P. partitions only, and G_1 - the class of all automata with one nontrivial S.P. partition only.

From Proposition 1.2 and results of /9/ if follows that if for some n the class G_n is nonempty then G_n contains also a strongly connected automaton, which is the direct product of strongly connected automata, i.e. strongly related automata. The notion of strongly related automata was first introduced in /12/. Still open is the problem of finding pairs (G,L) of classes of automata such that any two automata A,B are strongly related, where $A \in G$, $B \in L$. A partly solution of this problem can be found in /7,10/.

2. Total Automata.

The concept of a total automaton was introduced in /1/ and studied in detail in /2,3/. First we adopt some new definitions.

An automaton $A = (S,J)$ such that if for $s \in S$ and $f,f' \in \langle J \rangle$ we have $sf = sf'$ then $f \equiv f'$ is called state independent. An automaton $A = (S,J)$ is called total if and only if A is strongly connected, state independent, and $\langle J \rangle$ is a group. It is well known that for a total automaton $A = (S,J)$ there exists an operation over S such that S is a group, isomorphic to $\langle J \rangle$, and J is a set of right translations of S, corresponding to some set of generators of S. We say also that A is a total automaton over a group S.

All notations and definitions of group theory used but not explained here, can be found e.g. in /8/.

PROPOSITION 2.1. Let $A = (S,J)$ be a total automaton and the order of group S be equal to some prime number p. Then $A \in G_0$.

PROOF. Straightforward.

PROPOSITION 2.2. Let $A = (S,J)$ be a total automaton and the order of group S be equal to pq, where p,q are prime numbers, $1 < q < p$. Then either $A \in G_2$ or $A \in G_{p+1}$.

PROOF. Any group S of order pq is either a cyclic group of order pq or a nonabelian group with two generators a and b such that $a^q = 1 = b^p$ and $ab = ba^r$, where q divides p-1, $r \not\equiv 1 \pmod p$ and $r^q \equiv 1 \pmod p$. In the first case S is generated by an element a, $a^{pq} = 1$, and there exist just two nontrivial S.P. partitions π_0 and π_1 on A defined as follows $1 \equiv a^p(\pi_0)$ and $1 \equiv a^q(\pi_1)$. For the second case, there exist p+1 nontrivial S.P. partitions $\pi_0, \pi_1, \ldots, \pi_p$ on A defined by $1 \equiv a(\pi_0)$, $1 \equiv ab(\pi_1)$, ..., $1 \equiv a^{p-1}b(\pi_{p-1})$, $1 \equiv b(\pi_p)$. In both cases all nontrivial S.P. partitions are incomparable.

PROPOSITION 2.3. Let $A = (S,J)$ be a total automaton and the order of group S be equal to p^2, where p is a prime number, $1 < p$. Then either $A \in G_1$ or $A \in G_{p+1}$.

PROOF. A group S or order p^2 is either a cyclic group generated by an element a, $a^{p^2} = 1$, or an elementary abelian group with two generators a and b, $a^p = b^p = 1$, $ab = ba$. In the first case for A there exists just one

nontrivial partition π defined by $1 \equiv a^p(\pi)$. For the second case there exist p+1 incomparable nontrivial S.P. paritions $\pi_0,\pi_1,\ldots,\pi_p$ on A defined by $1 \equiv a(\pi_0)$, $1 \equiv ab(\pi_1)$, ..., $1 \equiv a^{p-1}b(\pi_{p-1})$, $1 \equiv b(\pi_p)$.

PROPOSITION 2.4. Let A = (S,J) be a total automaton and the order of group S be different from p, pq or p^2, where p and q are prime numbers. Then L(A) contain comparable partitions.

PROOF. The proof is lengthy but straightforward and hence it will be omitted.

COROLLARY. Let A = (S,J) be a total automaton with incomparable nontrivial S.P. paritions. Then the number of blocks and the cardinality of any block of a nontrivial S.P. partition on A are prime numbers. Moreover, all blocks of a nontrivial S.P. partition on A have the same cardinality.

3. Incompletely Specified Automata.

An incompletely specified automaton A is a pair (S,J) where S is a finite nonempty set and J is a nonempty set of incompletely specified functions $S \to S$, i.e. we admit here that members of J are not "into" functions. For the incompletely specified automaton A, a partition π on S has the S.P. if and only if $s \equiv s'(\pi)$ for $s,s' \in S$ and for any $f \in J$ with both sf and s'f specified implies that $sf \equiv s'f(\pi)$.

Let us consider the toal automaton A = (S,J) over group S with two generators a and b, $a^p = b^p = 1$, ab = ba, $|S| = p^2$, where p is a prime number, $1 < p$. Let B be an incompletely specified automaton $B = (S, J \cup \{f_m\})$, where f_m is a function of $M = \{1,a,\ldots,a^m,a^mb\}$ into M, $m < p$, $f_m(1) = (a), f_m(a) = a^2, \ldots, f_m(a^{m-1}) = a^m$, and $f_m(a^m) = f_m(a^mb) = a^mb$. By a straightforward verification we see that the S.P. partitions on A, defined by $1 \equiv a^{m+1}b(\pi_{m+1})$, ..., $1 \equiv a^{p-1}b(\pi_{p-1})$, $1 \equiv b(\pi_p)$ remain S.P. partitions on B. Hence L(B) contains p-m incomparable nontrivial S.P. partitions. Thus we show the following result.

PROPOSITION 3.1. For any natural number n there exists an incompletely specified automaton with exactly n incomparable S.P. partitions.

REFERENCES

1. R. Bayer, Automorphism groups and quotients of strongly connected automata and monadic algebras. Rep.204, Dep. Computer Sci., U. Illinois, Urbana, May 1966.

2. R. Bayer, On endomorphisms and congruences of automata, Math. Note No. 497, Mat. Res. Lab., Boeing Sci. Res. Lab., 1967.

3. W. Brauer, Gruppentheoretische Untersuchungen bei endlichen Automaten, Z. Angew. Math. Mech. 48, (1968) T113-T115.

4. P. Deussen, Halbgruppen und Automaten, Springer-Verlag, Berlin-Heidelberg-New York, 1971.

5. P. Goralcik, Charles U., Prague, Czechoslovakia. Private communication.

6. J.W. Grzymala-Busse, On the strongly related automata, Proc. Internat. Symp. "Discrete Systems", Riga, USSR, vol. 4, 118-127.

7. J.W. Grzymala-Busse, On the set of all automata with the same monoid of endomorphisms, Mat. Foundations Computer Sci., 4th Symp., Marianske Lazne, Czechoslovakia, 1975. Lecture Notes in Computer Sci., vol. 32, Springer-Verlag, Berlin-Heidelberg-New York, 1975, 246-251.

8. M. Hall, Jr., The Theory of Groups, The Macmillan Co., New York, N.Y., 1959.

9. J. Hartmanis, R.E. Stearns, Algebraic Structure Theory of Sequential Machines, Prentice-Hall, INc., Englewood Cliffs, N.J., 1966.

10. A.B. Miadowicz, and B. Mikolajczak, On the automorphism group of some strongly related automata and structural properties of finite automata extensions, Found. Control Engng. 1,2 (1976), 83-95.

11. E.T. Schmidt, Kongruenzrelationen algebraischer Strukturen, VEB Deutscher Verlag der Wissenschaften, Berlin, 1969.

12. G.P. Weeg, The automorphism group of the direct product of strongly related automata, J. Assoc. Comput. Mach. 12,2 (1965), 187-195.

Technical University of Poznan
60-965 Poznan
POLAND

and

University of Kansas
Lawrence, Kansas 66045
U.S.A.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

Алгоритмические вопросы программ синтеза, основанных на жесткой структуре

Б.Балог, А.Хорват, П.Леваи

НИИ Дальней Связи Будапешт

Программы, выработанные для автоматизирования задач синтеза в системе автоматического проектирования АУТЕР и излагаемые в докладе /LSINT-1, LSINT-2/ предполагают единую систему описания задач (схема процесса) и жесткую структуру осуществления (PLA, память типа D, постоянная память). Таким образом способы синтеза оказывают решение специальных случаев традиционных аггоритмических задач (кодирование состояний, минимализация).

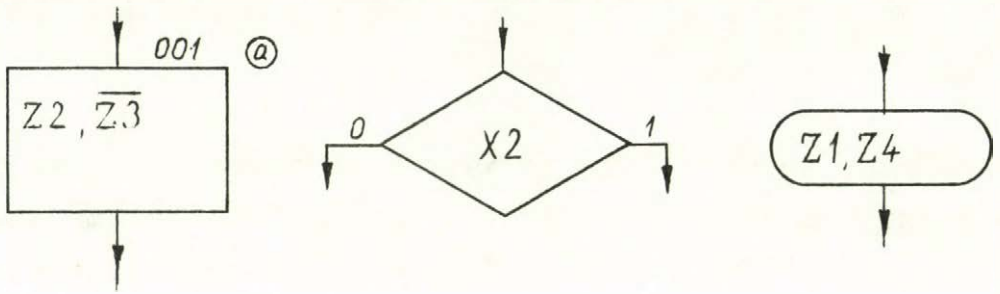
После краткого общего изложения, решение нескольких применяемых алгоритмов излагается примерами-образцами. Автоматический синтез цифровых схем осуществляется всегда с применением каких-то жестких структур. Стремление на всеобщность оказывается при методах описания, определяющих функцию схемы. Описания, самые подходящие к анализу и синтезу расходятся, хотя имеются методы которые одинаково хорошо применяются в обоих случаях.

Мы стремились разработать программы автоматического синтеза, которые осуществляют синхронные схемы управления с жесткой структурой и исходят из общего алгоритмического описания задачи.

Основными точками зрения, при выборе структур осуществления являлись применение компонентов большой сложности благоустроенное осуществление схемы, хорошая производимость и проверяемость.

Задание спецификации со структурной схемой

Система структурной схемы, выбранная для описания задачи содержит три элемента: элемент состояния, элемент решения и элемент условных выходов (Рис. I). Связью таких элементов описывается любая схема управления, имеющая модель типа Мили или Мур.



элемент состояния элемент решения элемент условных выходов

Рис. I.

Главным преимуществом этой системы структурной схемы является возможность обозримого описания задачи, хорошо обрабатываемого ЭВМ.

Структуры осуществления

С целью простого осуществления управляющих схем, сложные логические функции осуществляются в ЗУ. Применяемыми типами ЗУ, мы выбрали программируемую логическую матрицу (ПЛМ) и ПЗУ.

При применении ПЛМ, переменные состояния осуществляются памятями типа D. Таким образом, для каждой переменной состояния требуется только один выход ПЛМ. Принципиальная схема этого типа осуществления показана на рис. 2.

При применении памятей ПЗУ, мы выбрали микропрограммированную структуру, микрослово которой разделяется только гори-

зонтально и содержит так называемое "предлагаемое следующее поле адреса". Принципиальная схема этого видна на Рис. 3.

Рис. 2. Реализация с ПЛМ

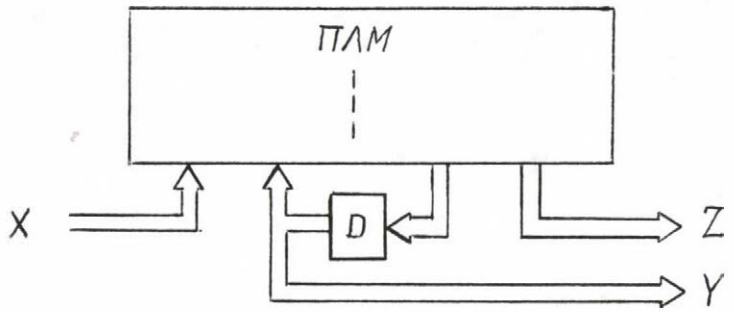
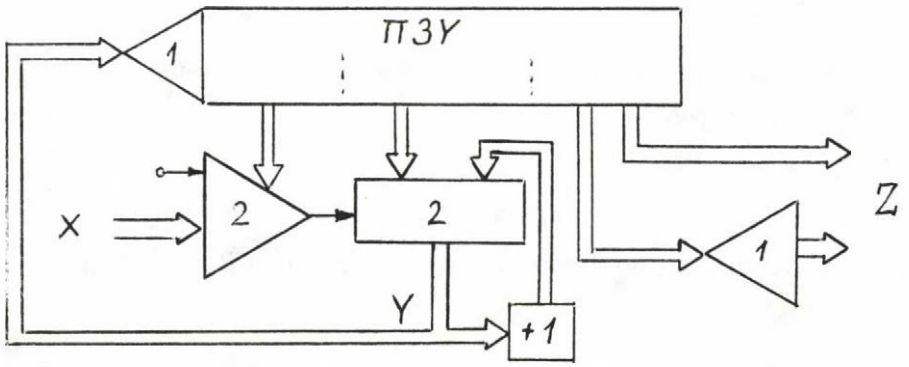


Рис. 3. Реализация с ПЗУ



I : декодер
2 : селектор

Задачи программ синтеза

Программа синтеза, применяющая ПЛМ / 2 / требует описание задачи с диаграммой процесса, предписания на кодирование состояний и управляющие данные, влияющие на процесс и результат синтеза. На основе этого, она исполняет кодирование состояний и упрощение или минимизацию функций. Как результат, получаются данные, программируемые с ПЛМ.

Программа основана на осуществлении ПЗУ и состоит из двух частей / 3 /.

В первой фазе, при кодировании состояний необходимо принимать **дальнейшие** состояния к состояниям структурной схемы (на **основе исследования** функций многих переменных, описывающих **переходы состояний** и наконец, вследствие перехода к модели **Мура. Дальнейшей** задачей является поиск максимальных совместимых **групп** кодированных выходных функций.

В **докладе выбираются** две из вышеуказанных алгоритмических задач, **с целью** изложения решения возникнутых интересных проблем.

Кодирование состояний

При кодировании, задачей является назначение к состояниям величины переменных состояния, чтобы получить оптимальную по стоимости функцию управления. Определение стоимости зависит от технологии. На уровне вентилей, стоимость обыкновенно считывается как число вентилей, или число входов вентилей (число диодов). При осуществлении функции как сумма конъюнкций, стоимость = число конънкций + число переменных в конъюнкциях.

В случае ПЛМ число переменных в конъюнкции не имеет влияние на число требуемых компонентов, так стоимость лучше выражается числом конъюнкций, т.е. числом строк в ПЛМ. Настоящая стоимость (в первоначальном значении слова) зависит только от числа применяемых компонентов, и так независима от числа применяемых строк в ПЛМ. Вследствие этого алгоритмы кодирования и упращения функции не должны быть оптимальными в каждом случае, т.е. два алгоритма, имеющие в результатах разное число строк ПЛМ, но одинаковое число компонентов, считаются равными.

Выходы ПЛМ осуществляют выходы схемы и управляющие функции памятей. Число выходов зафиксировано при данном типе ПЛМ. Поэтому мы применяем памяти типа D, имеющие только один вход.

Выбранный алгоритм кодирования базируется на осуществлении компонентами ПЛМ и памяти типа D. Результатом обработки структурной схемы функции перехода состояний создаются как суммы конъюнкций. Эти функции упрощаются по одиночке алгоритмом, излагаемым в докладе.

После этого шага создается матрица, содержащая число конъюнкций в функциях перехода состояний. Каждый элемент E_{ij} матрицы соответствует числу конюнкций функции перехода $S_i \longrightarrow S_j$. Цель алгоритма - уменьшить число элементов матрицы.

Алгоритм применяет два свойства:

а) Выбор состояния кодом 0. При применении памятей типа D, функции перехода, ведущие в состояние с кодом 0, не надо осуществить. Из-за этого код 0 целесообразно связывать с состоянием, куда многие переходы ведут с многими кон юнкциями.

б) Сокращенная зависимость. Пусть $\left\{S_j^i\right\}$ $j=1,\ldots,k$ множество состояний, куда имеется переход из состояния S^i.
Принимаем логическое произведение кодов этих состояний по битам.
Если код одного из состояний $\left\{S_j^i\right\}$ равен этому произведению, например код состояния S_k^i, тогда функцию, описывающую переход $S^i \longrightarrow S_k^i$ можно заместить одной конъюнкцией (т.е. кодом состояния S^i).

Алгоритм кодирования поищет систему кодов, отвечающую требованиям потребителя и применяя вышеуказанные свойства, эффективно уменьшает число строк ПЛМ.

Минимизация программируемой логической схемы (ПЛС)

Под минимизацией ПЛС понимается решение следующей задачи: установить одну из дизъюнктивных нормальных форм данной Булевой функции, в которой число разных конъюнкций минимально. Мы решили эту задачу при многовыходной, неполно опреде-

ленной, данной своими импликантами Булевой функции, проектируя программу на малую ЭВМ.

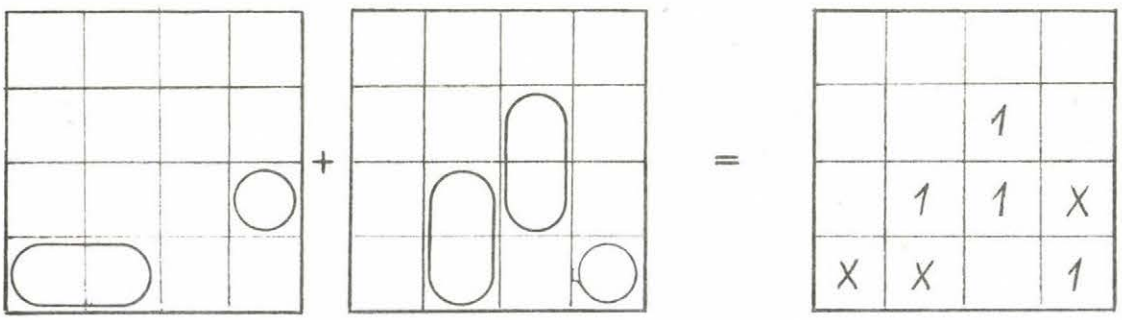
Нужно познакомиться с несколькими понятиями. Основные понятия предлагаются известными.

Рассмотрим одну Булевую функцию к-выходов. Под вектором неопределенных выходов (ВНВ) одной конъюнкции, мы понимаем тот двоичный вектор к-компонентов, i-ый компонент которого равен I-му, тогда и только тогда, если i-ый выход функции неопределен при данной конъюнкции. Относительно этих выходов, конъюнкция называется неопределенным импликантом (НИ).

Под вектором определенных выходов (ВОВ) одной конъюнкции, мы понимаем тот двоичный вектор к-компонентов, i-ый компонент которого равен I-му, тогда и только тогда, если i-ый выход функции равен I-му при данной конъюнкции. Относительно этих выходов, конъюнкция называется определенным импликантом (ОИ).

На рис. 4 виден пример, как дать функцию импликантами

Рис. 4. Данная импликантами Булева функция



Неопределенные импликанты

Определенные импликанты

ОИ называется приимпликантом (ПИ), если нет другой конъюнкции К, которая геометрически (т.е. в Булевом пространстве) заключает в себе ОИ, и $ВОВ_i$ (ОИ)=1 → $ВОВ_i$ (К)=1, i=1, 2,...k, где $ВОВ_i$ ( ) i-ый компонент соответственного вектора.

Любая Булева функция имеет такую минимальную дизъюнктивную нормальную форму, конъюнкции которой являются примимпликантами, значит довольно искать такую форму.

Две конъюнкции К1 и К2 сцепляются, если они удовлетворяют следующим условиям:

а) конъюнкция К3=К1∧К2 не равно тождественному нулю;

б) есть такой компонент i, $1 \leq i \leq k$, на которой

$$ВОВ_i/К1/ = ВОВ_i/К2/ = ВОВ_i/К3/ = 1$$

Последовательность ПИ1, ПИ2....ПИℓ называется соединяющей ПИ1 и ПИℓ цепью, если при любом индексе i, $1 \leq i \leq \ell - 1$ ПИi и ПИ(i+1) сцепляются. Какое-нибудь множество ПИ-ов называется связным, если для любых двух его ПИ-ов существует цепь, соединяющая их.

Если два ПИ-а нельзя соединить цепью, они независимы, с точки зрения покрытия функции. Этот факт дает нам возможность для разделения Булевой функции на максимальные связанные функции и для минимизации этих функций по одному.

ПИ1 доминирует ПИ2, если считая ПИ1 неопределенным импликантом, все компоненты ВОВ(ПИ2) становятся нулем.

ПИ1 и ПИ2 эквивалентные, если ПИ1 доминирует ПИ2 и обратно.

ПИ скажем существенным (СПИ), если считая все сцепляющиеся и неэквивалентные с ним ПИ неопределенным импликантом, в его ВОВ остается компонент, равный I-му.

Ясно, что в минимальной форме все классы эквивалентных СПИ надо репрезентировать и не выступает такой ПИ, который доминирован другим ПИ.

Мы называем упращающим процессом тот процесс, в течение которого уничтожаем доминированные ПИ-ы и ищем СПИ, которые все попадают в минимальную форму.

Схема минимизированной программы видна на рис. 5.

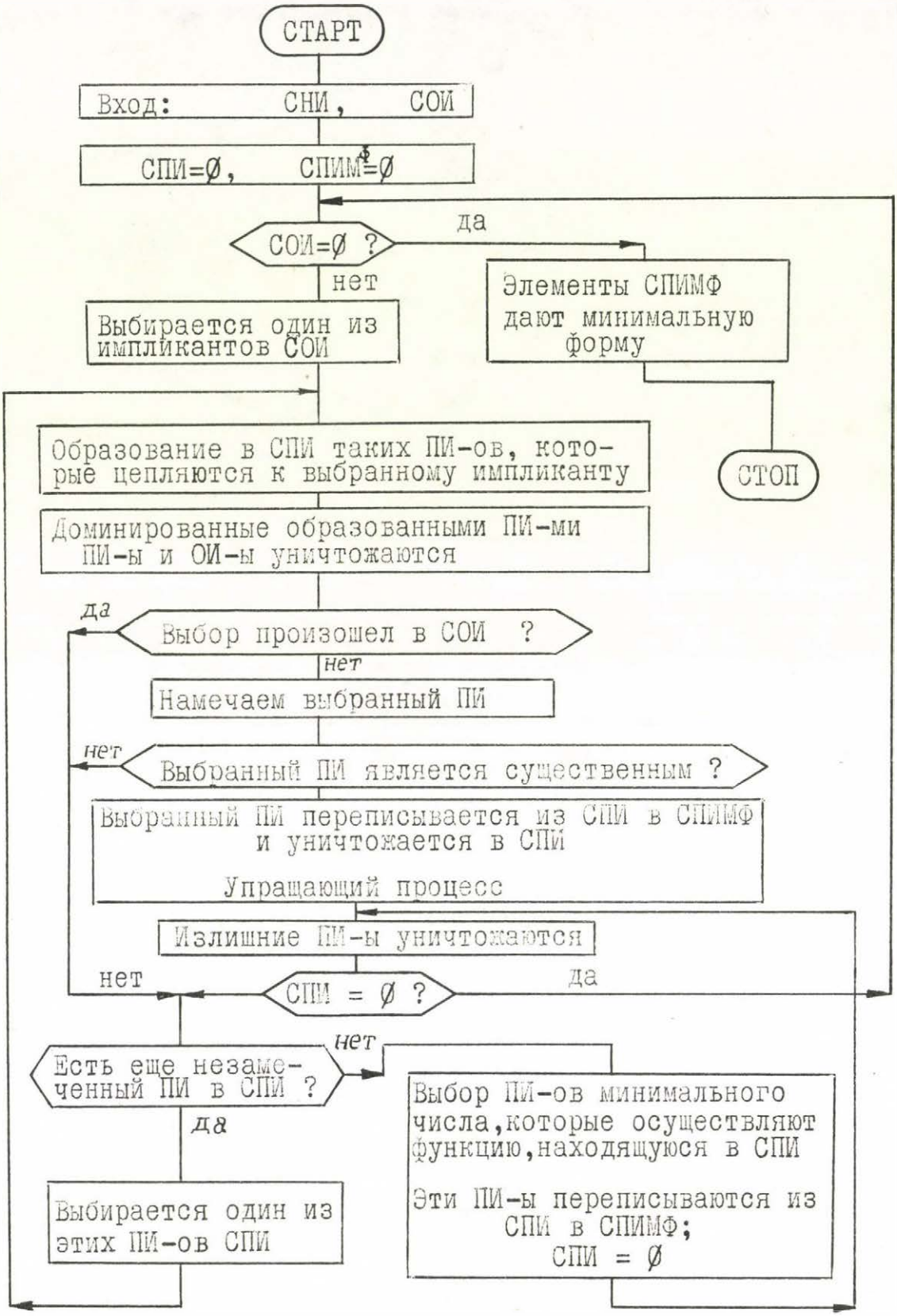


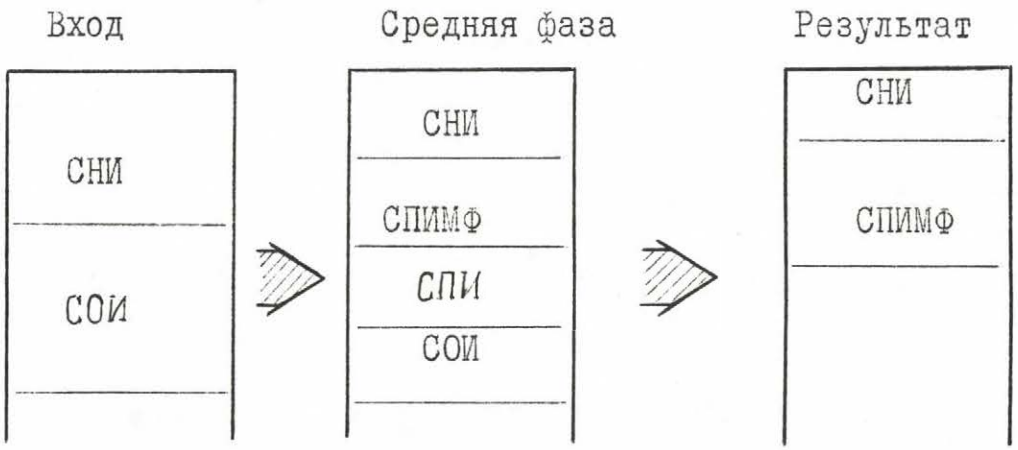
Рис. 5. Схема минимизации

Здесь СНИ и СОИ - список неопределенных и определенных импликантов (это входные списки), СПИ - список примимпликантов, СПИМФ - список примимпликантов минимальной формы.

При определении ВОВ и ВНВ одной конъюнкции, ПИ-ы СПИМФ надо считать неопределенными.

Изменение структуры данных в течение действия программы видно на рисунке 6.

Рис. 6. Изменение структуры данных



Мы считаем преимуществами программы следующие:

1° Связанные части функции минимизированы автоматически по одной.

2° Образование ПИ-ов является направленным, т.е. в то же время (точнее в одном цикле) образовываются только те Пи-ы, которые цепляются к тому же импликанту.

3° В каждом цикле делается попытка уменьшить число ПИ-ов упращающим процессом.

4° При образовании ПИ-ов, излишние элементы СОИ уничтожаются.

5° Если один ПИ попадает в СПИМФ, излишние элементы СНИ уничтожаются.

6° Если одна Булева функция при управляющем процессе дальше не упращается, тогда ПИ-ы минимального числа выбираются, принимая метод типа ветвей и границ. Мы ускоряем этот метод, принимая и здесь упращающий процесс.

7° В течение минимизации определение векторов выходов разных конъюнкций стало основной задачей. Мы решили эту задачу непосредственно с помощью данных импликантов, без разбития конъюнкции на элементарные конъюнкции.

Эти преимущества дали возможность для эффективности программы минимизации ПЛС-ы и на маленькой ЭВМ.

Список литературы

/ 1 / C.R. Clare: Designing Logic Systems Using State Machines McGraw-Hill Book Company 1973.

/ 2 / B. Balogh, A. Horváth: LSINT-1 felhasználói dokumentáció TKI 1978. /внутренняя публикация/

/ 3 / B. Balogh, A. Horváth: LSINT-2 felhasználói dokumentáció TKI 1979. /внутренняя публикация/

/ 4 / T. Rhyne et al.: A new technique for the fast minimization of switching functions IEEE Tr. Comput. 1977/8.

/ 5 / E. Morreale: Recursive operators for prime implicant and irredundant normal form determination IEEE Tr. Comput. 1970/6.

/ 6 / N. Christofides: Graph Theory - an algorithmic approach Academic Press 1975.

MTA Számitástechnikai és Automatizálási Kutató Intézete Tanulmányok 137/1982

Pásztorné Varga Katalin

НА ОСНОВАНИИ БИНАРНЫХ ОТНОШЕНИЙ УПОРЯДОЧЕНИЕ В КЛАСС ЭКВИВАЛЕНТНОСТИ ЭЛЕМЕНТОВ МНОЖЕСТВА С ПРИМЕНЕНИЕМ ТЕХНИКИ СТЕКОВ

Исследовательский Институт Вычислительной Техники и Автоматизации Венгерской Академии Наук

0. ВВЕДЕНИЕ

В различных процессах поиска, в трансляторах, в синтактических проверках проблемы часто сводятся к задаче прохождения дерева. При реализации различных типов прохождения дерева стековая техника является привычным и эффективным методом. В статье решение одного типа проблемы сводится к задаче прохождения дерева и для реализации его применяется техника стеков. Проиллюстрируем применение метода решением задач, вытекающей из некоторых проблем области проектирования вычислительной машиной.

I. Графы и бинарные отношения

Пусть задано бинарное отношение R на множестве H . Известно, что всегда возможно представить R с помощью такого ориентированного графа G_R вершины которого соответствуют элементам H и от вершины P к вершине Q ведет дуга, тогда и только тогда, если PRQ

Как известно, если R

– отношение эквивалентности, то G_R такой неориентированный граф, который состоит из изолированных полных графов, соот-

ветствующих классу эквивалентности H (рис.I.);

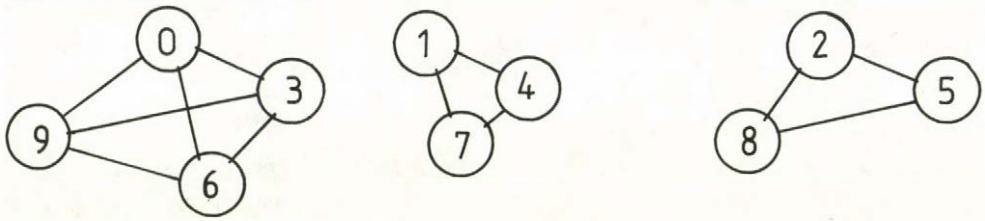


рис. 1

– отношение упорядочения, то G_R сриентированный граф без цикла. Одновременно G_R является диаграммой сети порождающей R (рис.2.);

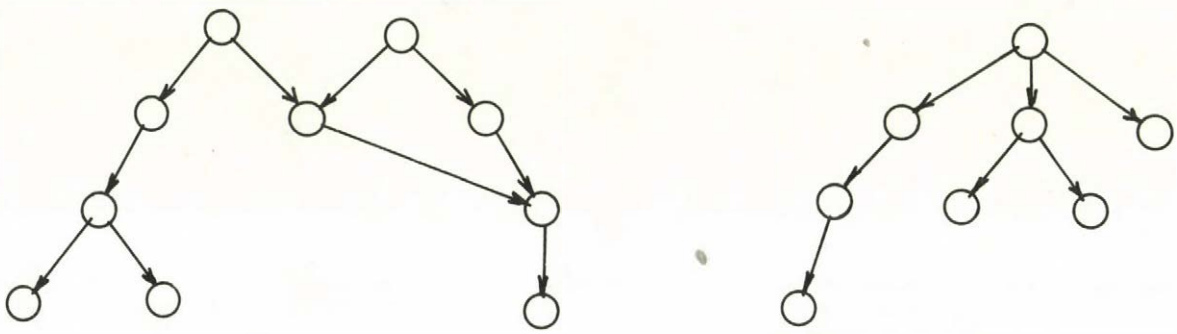


рис. 2

– любое бинарное отношение, тогда G_R обыкновенный ориентированный граф с циклом (рис.3.).

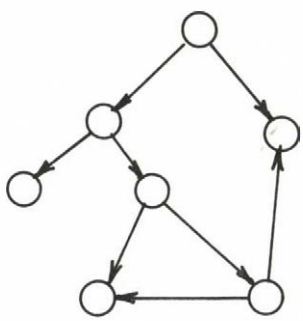


рис. 3

2. Формулировка задачи

Пусть задан язык L(G), G=(T,N,M,SZ) где

T – множество терминальных символов,
N – множество нетерминальных символов,

M – символ предложения,

SZ – множество правил постановок.

Пусть будет $H=\{M_j\}\in L(G)$ и заданы бинарные отношения R_T, R_E, R_H, R

R_T и R_E определены на T

- R_T отношение упорядочения, по которому элементы T упорядоченены в последовательность,
- $R_E(R_T)$ отношение эквивалентнссти,
- R_H и R определены на H,
- $R_H(R_T, R_E)$ бинарное отношение,
- R отношение эквивалентности, для которых $M_iR_HM_j \rightarrow M_iRM_j\ (M_i, M_j \in H)$, то есть R_H имплицириет R.

Задача – упорядочить в класс эквивалентности элементов H по R на базе R_H.

3. Решение задачи

Так как R_H известно, то известно и G_{R_H}. Ввиду того, что $R_H \rightarrow R$, где R отношение эквивалентности G_{R_H} не сильно связанный и вершины связанных подграфов задают классы эквивалентности по R. Перечисление вершин этих подграфов дает решение задачи.

Если мы знали бы покрывающее дерево подграфов, то алгоритмы для получения классов эквивалентности было бы следующим:

1. Элементам H приписываем индексы $H=\{h_1, h_2, \dots, h_n\}$
2. Если уже посмотрели все элементы H, то алгоритм кончается.
3. $i=j$, где $h_j \in H$ неразработанный корень с минимальным индексом.
4. Прохождение дерева, имеющего корни h_i. Наименование вершин, составляющих дерево, соберем в одну группу. Если во время сборки найдется вершина, принадлежащая к другой группе, то эти группы объединяются.
5. Переход к пункту 2.

С точки зрения решаемой задачи сказалось целесообразным прямой порядок прохождения дерева. Из-за особенностей задачи был модифицирован алгоритм прямого порядка прохождения. В модифицированном алгоритме в стек записываются терминальные вершины дугов, исходящих из вершин разветвления вместо вершин разветвления.

В дальнейшем опишем модифицированный алгоритм прямого порядка прохождения.

Обозначения:

K – корень дерева
AC – исследуемая вершина
STACK – идентификатор стековой памяти
TREE – массив переменных для перечисления вершин графа
IS – актуальный индекс STACK
IT – актуальный индекс TREE
LNEXT(A) – терминальная вершина дуги, исходящей влево из вершины AC (рис.4)
RNEXT(A,I) – терминальная вершина i-той дуги, исходящей вправо из вершины AC (рис.4)
ACOUT – число дуг, исходящих из исследуемой вершины AC

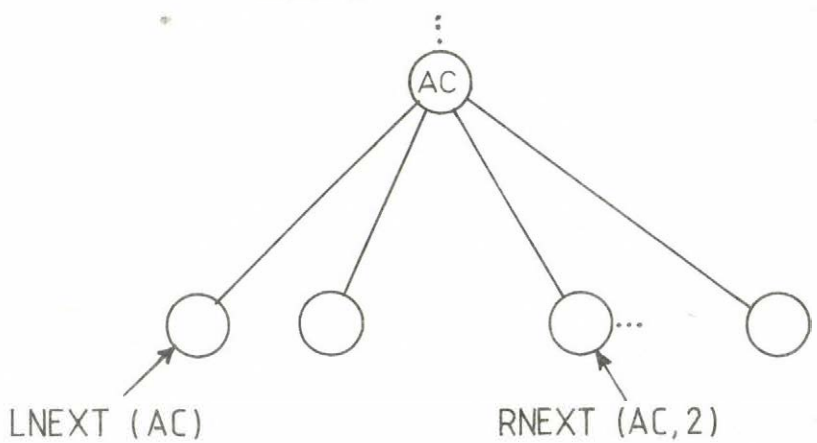


рис. 4

Алгоритм

1. IS=0
2. IT=IT+1, TREE(IT)=AC
3. Если AC не является листом, то переход к.п.6.
4. Если IS=0 , то TREE(1) - TREE(IT) содержат все вершины дерева. Стоп.
5. AC=STACK(IS), IS=IS+1 , Переход к.п.2.
6. Если ACOUT=1 , то переход к.п.8.
7. (STACK(IS+I)=RNEXT(AC,I) I=1,ACOUT-1) IS=IS+ACOUT-1
8. AC=LNEXT(AC). Переход к.п.2.

Сильно связанный ориентированный граф не всегда покрываемый одним остовым деревом. В этом случае вершины, относящиеся к двум остовным деревам нужно объединить в одну группу, если они имеют хотя бы одну общую вершину.

Высший алгоритм может быть приспособен на то, чтобы знав только ориентированный граф, он выбирает и проходит основное дерево подграфа, достижимого из одной вершины графа, и он объединяет принадлежащие одной группе вершины. Для этого достаточно заменить 7-й и 8-й пункты.

7'a. Если RNEXT(AC,I) уже принадлежит одной группе, то объединяем эту группу с группой, полученной в данной фазе прохождения. Переход к 7'c.

7'b Если RNEXT(AC,I) находится уже в STACK -е, то не занимаемся с ней, иначе IS=IS+1, STACK(IS)=RNEXT(AC,I).

7'c. Если I<ACOUT-1, то I=I+1. Переход к 7'a.

8'a. Если LNEXT(AC) уже принадлежит одной группе, то объединяем эту группу с группой, полученной в данной фазе прохождения. Переход к 4.

8'b Если LNEXT(AC) находится уже в STACK -е, то не занимаемся с ней. Переход к 4. Иначе AC=LNEXT(AC), Переход к 2.

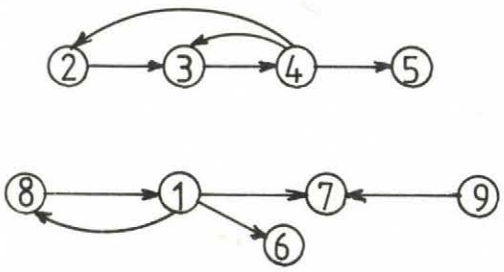
Алгоритм выбора связанных подграфов произвольного ориентированного графа.
Пусть будет множество вершин графа $H=\{h_1, h_2, \cdots, h_n\}$

1. Выбор еще не обработанного элемента h_i с минимальным индексом.
2. Прохождение подграфов, достижимых из h_i (модифицированный алгоритм прохождения). Отметим достижимые h_i
3. Если все элементы $h_i \in H$ отмечены, то алгоритм закончен, иначе переход к п.1.

Пример для иллюстрации функционирования алгоритма:

$H=\{1,2,3,4,5,6,7,8,9\}$

R_H:



С помощью алгоритма

- выбирается элемент $1 \in H$
- осуществляется прохождение дерева 1(8,6,7) и упорядочение элементов 1,6,7,8 в одну группу.
- результат отметки: $H=\{(1),2,3,4,5,(6),(7),(8),9\}$
- выбирается элемент $2 \in H$
- осуществляется прохождение дерева 2(3) и упорядочение элементов 2,3 в одну группу.
- результат отметки: $H=\{(1),(2),(3),4,5,(6),(7),(8),9\}$
- выбирается элемент $4 \in H$

- осуществляется прохождение дерева 4(3,5) упорядочение (из-за элемента 3) элементов 2,3,4,5 в одну группу.
- результат отметки: $H=\{①,②,③,④,⑤,⑥,⑦,⑧,⑨\}$
- выбирается элемент $9 \in H$
- осуществляется прохождение дерева 9(7) и упорядочение элемента 9 в одну группу, которая содержит элемент 7.
- результат отметки - отмечены все элементы H .

Классы эквиваленции: (1,6,7,8,9), (2,3,4,5)

Алгоритм завершен.

4. Задачи применения

Известно описание участков проводников ТЭЗ. Задача собирать в одну группу участки проводников, которые соединяются на ТЭЗ.

В нашем случае $L(G)$ - множество всех возможных участков проводников h_i ТЭЗ и $H \leq L(G)$
В грамматике $G = T,N,M,SZ$:

T - множество точек $t_i(x_i, y_i)$, где $0 \leq x_i \leq N_x$
$0 \leq y_i \leq N_y$ целые числа

M - символ предложений

N - $\{M_n\}$, $n=1,2,\ldots$

SZ - совокупность правил

$M_n \rightarrow t_{i_1} t_{i_2} \ldots t_{i_n}$

$M \rightarrow M_n$

$M \rightarrow M_n t_{i_n} t_{i_{n+1}}$

$M \rightarrow t_j t_{i_1} M_n$

R_T - отношение упорядочения в T , по которому элементы T упорядочимы по координате x.

$R_E(R_T)$ - отношение эквивалентности в T .
$t_i R_E t_j$, если $y_i = y_j$

$R_H(R_E, R_T)$ - двоичное отношение в H. $h_i R_H h_j$, если

$$(\exists t_i t_{i+1} t_j)(t_i, t_{i+1} \in h_i \;\&\; t_j \in h_j \;\&\; t_i R_T t_{i+1} \;\&\; t_i R_E t_j \;\&\; t_i R_T t_j \;\&\; t_i R_T t_{i+1})$$

R – отношение эквивалентности в H $h_i R h_j$, если h_i и h_j гальванически связаны.

Очевидно, что $R_H \to R$

Ввиду того, что знание R_H эквивалентно знанию G_{R_H} в принципе предыдущий алгоритм алгоритм применим. Как мы уже видели, алгоритм переработает каждый элемент H только один раз. Поэтому достигать дальнейшее улучшение эффективности алгоритма таким образом не возможно. Эффективность алгоритма может быть улучшена в ходе вычисления функций LNEXT и RNEXT т.е. решение о существований R_H . В данном случае нам удалось достичь дольнейшее улучшение с формулировкой $R_H(R_T, R_E)$ в виде $R_H(R_T)$ и все h_i встречающиеся в H были физически упорядочены по R_T .

Обозначения в алгоритме, решающий задачу:

IS – индекс массива STACK

IT – индекс массива TREE

IR – второй индекс массива RNEXT

Q – рабочая переменная

CONN – порядковый номер класса эквивалентности, находящийся под обработкой

IC – номер классов эквивалентности

CLASS(I) – место элементов I–го класса эквивалентности

$$H - \{h_1, h_2, \ldots, h_n\}$$

$$h_i - t_{i_1} t_{i_2} \ldots t_{i_{n_i}}$$

Алгоритм

1. J=0, IC=0
2. Если все $h_i \in H$ уже обработаны, то – стоп. Иначе выбирать минимальный индекс i , для которого h_i еще не обработанный. J=i, AC=h_i, IS=IT=CONN=0.
3. IT=IT+1, TREE(IT)=AC.
4. Если AC лист, то переход к п.16.

5. Если ACOUT=1 , то переход к п.12.

6. IR=1.

7. Q=RNEXT(AC,IR)

8. Если Q ∈ CLASS(I), I≤IC , то TREE(1)-TREE(IT)→CLASS(I), CONN=I, IT=0. Переход к п.11.

9. Если Q ∈ STACK , то переход к п.11.

10. IS=IS+1, STACK(IS)=Q

11. Если ACOUT-1>IR , то IR=IR+1. Переход к п.7.

12. L=LNEXT(AC)

13. Если L ∈ CLASS(I), I≤IC то TREE(1)-TREE(IT)—CLASS(I), CONN=I, IT=0. Переход к п.16.

14. Если L ∈ STACK, то переход к п.16.

15. AC=1. Переход к п.3.

16. Если IS≠0 то AC=STACK(IS), IS=IS-1. Переход к п.3.

17. Если CONN=0, то IC=IC+1, CONN=IC.

18. TREE(1)-TREE(IT) → CLASS(CONN). Переход к п.2.

СПИСОК АВТОРОВ

СПИСОК ВЫШЕДШИХ СБОРНИКОВ ДОКЛАДОВ СЕМИНАРОВ И СИМПОЗИУМОВ НКС СЭВ ПО ТЕМЕ 1-15.1 В СЕРИИ **MTA SZTAKI TANULMÁNYOK**

№ 21 Доклады семинара, проводимого 19-23-ого февраля 1973 г. в Венгрии во время совещания по теме 1-15.1. "Структура и методологические рекомендации по созданию систем автоматизированного проектирования дискретных устройств". Будапешт, 1974.

№ 63 Доклады семирана, проводимого 20-21 мая 1976 г. в Риге во время совещения по теме 1-15.1. "Разработка общей теории автоматов".
Будапешт, 1977.

№ 99 Доклады симпозиумов, проводимых 18-23 ареля 1977 года в Яхранке и 8-13 мая 1978 года в Доновали во время совещаний НКС СЭВ по теме 1-15.1. "Теория автоматов в применении к проектированию дискретных устройств и систем". Будапешт, 1980.

№ Доклады симпозиумов, проводимых 23-28 апреля 1979 года в Вайсинге и 5-10 мая 1980 года в Вишеграде во время совещаний НКС СЭВ по теме 1-15.1- "Теория автоматов и ее приложения к проектированию дискретных устройств и систем".
Будапешт, 1982. /завершающий том/

A TANULMÁNYOK SOROZATBAN 1982-BEN MEGJELENTEK:

130/1982 Barabás Miklós - Tőkés Szabolcs:
A lézer printer képalkotás hibái és optikai korrekciójuk

131/1982 RG-II/KNVVT "Szisztemü upravlenija bazani i informacionnüe szisztemü" Szbornik naucsno-iszszledovatel'szkih rabot rabocsej gruppü RG-II KNVVT, Bp. 1979. T o m I.

132/1982 RG-II/KNVVT T o m I I .

133/1982 RG-II/KNVVT T o m I I I.

134/1982 Knuth Előd - Rónyai Lajos: Az SDLA/SET adatbázis lekérdező nyelv alapjai /orosz nyelvü/

135/1982 Néhány feladat a tervezés-automatizálás területéről. Örmény-magyar közös cikkgyüjtemény

136/1982 Somló János: Forgácsoló megmunkálások folyamatainak optimálási és irányitási problémái

Főprint nyomda 82207